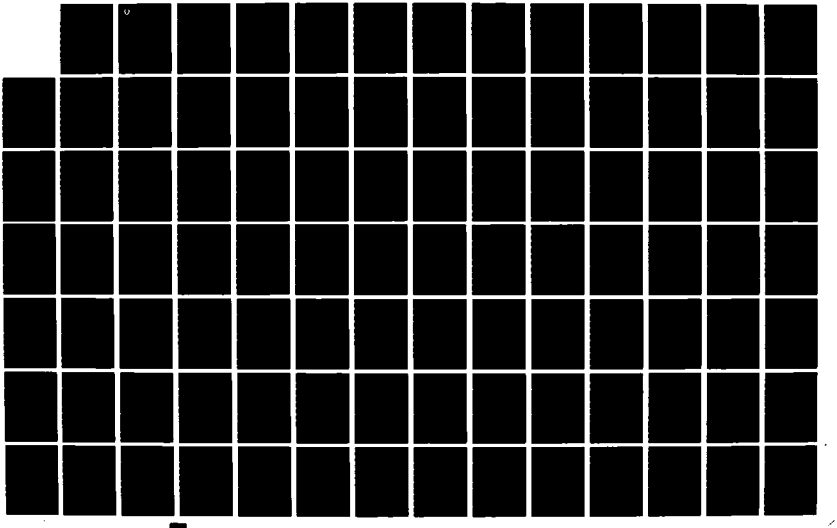
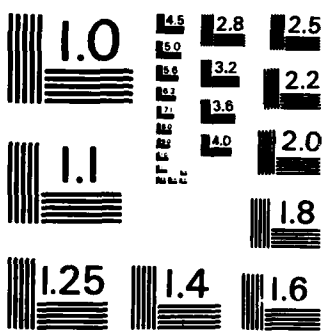
AD-A132 893 DDCENTRALIZED CONTROL OF SCHEDULING IN DISTRIBUTED SYSTEMS(U) MASSACHUSETTS UNIV AMHERST DEPT OF ELECTRICAL AND COMPUTER ENGINEERING J A STANKOVIC 1/2

UNCLASSIFIED 18 MAR 83 DAAB07-82-K-J015 F/G 9/2 NL





RESEARCH AND DEVELOPMENT TECHNICAL REPORT
CECOM

DECENTRALIZED CONTROL OF SCHEDULING IN DISTRIBUTED SYSTEMS

JOHN A. STANKOVIC

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
UNIVERSITY OF MASSACHUSETTS
AMHERST, MA 01003

18 March 1983

5th QUARTERLY REPORT FOR PERIOD 15 DEC 82 to 4 MARCH 83

Distribution Statement

Approved for public release;
distribution unlimited.

PREPARED FOR
CENTER FOR COMMUNICATIONS SYSTEM (C. GRAFF)

CECOM

U S ARMY COMMUNICATIONS-ELECTRONICS COMMAND
FORT MONMOUTH, NEW JERSEY 07703

CLEARANCE OF TECHNICAL REPORTS FOR PUBLIC RELEASE

(AR 360-5, AR 70-31 and AR 530-1) SUBMIT IN DUPLICATE

TO: ERADCOM Tech Spt Actv DELSD-L-S	FROM: DRSEL-COM-RF-2 Signal Processsing Division	DATE: 13 May 1983

INSTRUCTIONS: SECTION I: Self explanatory. SECTION IIa: Self explanatory; b, c, d, & e: If the report contains any of this information, Distribution Statement B applies (see DOD Dir 5200.20; f & g: Delete this information from report; h,i & j: Coordinate and obtain approval from agencies concerned. SECTION III: Self explanatory. SECTION IV: Self explanatory.

In compliance with AR 360-5, AR 70-31 and AR 530-1 the attached unclassified technical report is forwarded for clearance for public release and assignment of Distribution Statement A: "Approved for public release; distribution unlimited."

SECTION I DESCRIPTION

1. TITLE OF REPORT
Decentralized Control of Scheduling in Distributed Systems

2. AUTHOR(S) EXT NO
Dr. J. Stankovic

FOR CONTRACT REPORTS

3. CONTRACTING OFFICER'S TECHNICAL REPRESENTATIVE COTR Charles J. Graff EXT NO 65619

4. CONTRACTOR Dept. of Electrical Engineering, University of Mass.

5. CONTRACT NO DAAB07-82- K-J015 REPORT NO 5th Quarterly

6. CONTRACTOR ☐ IS ☒ IS NOT AUTHORIZED ACCESS TO CLASSIFIED MATERIAL UNDER THIS CONTRACT

SECTION II BASIS FOR RELEASE

The report identified above has been reviewed by the undersigned who affirm that the document does not contain any of the following information:

a. Classified information.
b. Information furnished by a foreign government with the understanding that it not be released outside the U. S. Government.
c. Proprietary information for which authority for release has not been obtained.
d. Information resulting from test or evaluation of commercial products or military hardware.
e. Management reviews, records of contract performance evaluation, or other advisory documents evaluating programs of contractors.
f. Information that would be prejudicial or embarrassing to the Government, a foreign government, a contractor, a corporation or an individual.
g. Information concerning subjects of potential controversy among the military services.
h. Subject matter concerning significant policy within the purview of other agencies.
i. Subject matter that implies official positions or scientific attitudes of higher authority.

j. Subject matter on space, satellite, atomic or CW/BR activities requiring clearance of agencies concerned.

SECTION III OPSEC CLEARANCE

In addition, the undersigned certify that the attached R&D technical report has been cleared for public release in accordance with the OPSEC Plan of this activity as prescribed by AR 530-1 and DARCOM supplement 1 thereto.

John E. Quigley
JOHN E. QUIGLEY
OPSEC OFFICER

John E. Quigley
for CHARLES J. GRAFF
AUTHOR (OR COTR)

James H. Salton
JAMES H. SALTON, Acting Chief
Signal Processing Division
DIVISION DIRECTOR

SECTION IV CLEARANCE

TO:	FROM: ERADCOM TECH LIBRARY, TSA STINFO DELSD-L-S	DATE:

The report identified in Section I is cleared for public release. It may be made available to the National Technical Information Service. It may also be presented at symposia or published in the open literature in the United States without further clearance, provided no change is made contravening the basis of release certified to in Sections II and III.

CHIEF, STINFO OFFICE

NOTICES

Disclaimers

The citation of trade names and names of manufacturers in this report is not to be construed as official Government indorsement or approval of commercial products or services referenced herein.

Disposition

Destroy this report when it is no longer needed. Do not return it to the originator.

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Decentralized Control of Scheduling in Distributed Systems

5. TYPE OF REPORT & PERIOD COVERED

5th Quarterly Report
Dec 15, 1982 - March 14, '83

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

Dr. John A. Stankovic

8. CONTRACT OR GRANT NUMBER(s)

DAAB07-82-K-J015

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Dept of Electrical and Computer Engineering,
Univ. of Mass., Amherst, Mass. 01003

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

1L1.61102.AH48.DF.01

11. CONTROLLING OFFICE NAME AND ADDRESS

CDR US Army CECOM
DRSEL-COM-RF-2
Fort Monmouth, NJ 07703

12. REPORT DATE

18 March 1983

13. NUMBER OF PAGES

102

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Decentralized Control, Scheduling, Network Management.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This fifth quarterly report includes two technical papers. The first paper deals with three algorithms that could be utilized to control tasks/job scheduling in distributed systems. Using a simulation approach, insight into the performance and stability of decentralized job control scheduling algorithms is demonstrated. The second paper provides a survey of current research directions in distributed systems software. Research in distributed operating systems, languages and data bases is covered, and important research questions for distributed system software are posed.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Fifth
Quarterly Report

Professor John A. Stankovic
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, Massachusetts 01003

Contract Number: DAAB07-82-K-J015

Date of Report: March 18, 1983

Title of Report: Decentralized Control of Scheduling in Distributed Systems

Period: December 15, 1982 - March 14, 1983

Report prepared by: John A. Stankovic

Preface

This report includes an outline of the work performed during the fifth quarter of this contract and a budget summary. Two papers written during this quarter are added as appendices.

December 15, 1982 - January 14, 1983

Progress:

1. We wrote the annual report.

2. We converted the GPSS programs written earlier in the contract to run on the VAX for better accessibility, enhanced language features and reduced cost.

3. Each of the converted scheduling algorithm simulations were rerun to verify correctness.

4. Confidence intervals were generated for several of the algorithms by repeated simulations.

Results:

The simulations identified a simple but effective algorithm for decentralized load balancing. See the paper in the appendix entitled "Simulations of Three Adaptive, Decentralized Controlled Job Scheduling Algorithms."

Plans:

1. We plan to return to debugging the Dynamic Bayesian decision theory simulation program.

2. The two RA's will continue to design the simulation programs for the general bidding algorithm and the stochastic learning automata algorithm discussed in previous reports.

January 15, 1983 - February 14, 1983

Progress:

1. The Dynamic Bayesian decision theory simulation is completely debugged. Tests are being run and evaluated.

2. Both the bidding scheduling algorithm and the stochastic learning automata algorithms are designed and being implemented.

3. Additional work has begun in the area of scheduling with real time constraints.

Results:

Preliminary simulation results for the Dynamic Bayesian decision theory approach are being obtained.

Plans:

Continue all the above.

February 15, 1983 - March 14, 1983

Progress:

1. Simulations for Dynamic Bayesian decision theory are complete.

2. A paper on current research issues was written entitled "Current Research and Critical Issues in Distributed Software Systems" (see the appendix).

3. Implementation efforts described in the previous month are continuing.

Results:

Results of (1) above are being written in paper form. The first draft is almost complete.

Plans:

1. Continue the implementation of the simulations

2. Continue work on scheduling with real-time constraints.

2. Budget update for 2nd year of contract

Month	Planned
December (82)	6,809.53
January (83)	4,021.00
February	4,000.00
March	4,000.00
April	4,000.00
May	4,000.00
June	7,000.00
July	7,000.00
August	7,000.00
September	3,397.64
October	3,000.00
November	3,000.00
December	3,000.00
	55,830.53

Appendix

This appendix contains two papers produced as part of this contract. It is requested that both these papers be cleared for publication.

CURRENT RESEARCH AND CRITICAL ISSUES IN DISTRIBUTED SOFTWARE SYSTEMS

by

John A. Stankovic

Krithi Ramamritham

Walter H. Kohler

University of Massachusetts

Amherst, Massachusetts

Preparation of this material was funded in part by the following: the National Science Foundation under grants MCS 82-02586, MCS 81-04203, ECS 81-20931 and CECOM CENCOM grant DAAB07-82-K-J015.

ABSTRACT

A survey of current research in distributed systems software is presented. Emphasis is placed on research in distributed operating systems, programming languages for distributed systems, and distributed databases. While many references are made to existing research projects, systems and languages, an attempt was made to identify the important issues and, where possible, to categorize current solution techniques for the individual issues. Some of the more important open research questions for distributed systems software are itemized.

CR Classification

Categories and Subject Description

C. Computer Systems Organization

C.2. Computer-Communication Network
C.2.4 Distributed System
- Distributed Databases
- Network Operating Systems

D. Software

D.3 Programming Languages
D.3.3 Language Constructs
- Concurrent Programming Structures
- Control Structures

D.4 Operating Systems
D.4.1 Process Management
- Concurrency
- Deadlock
- Scheduling
- Synchronization

D.4.3 File Management Systems
- Distributed File Systems

D.4.4 Communication Managament
- Buffering
- Message Sending

D.4.7 Organization and Design
- Distributed systems

H. Information Systems

H.2 Database Management
H.2.4 Systems
- Distributed systems
- Transaction Processing

General Terms: Design, Language

Additional Keywords and Phrases: survey, research issues

1.0 INTRODUCTION

A distributed software system is one that is executed on an architecture in which multiple processors are connected via a communication network. The main issue in the design of such systems is: how are programs, data and control to be distributed among the components of a distributed system? In this paper we consider this issue from the perspective of the distributed operating system, programming language and database areas. The goal of this paper is to survey and integrate the current research in these areas as well as to identify some of the critical research problems for distributed software.

Section 2 summarizes current research in distributed operating systems and programming language areas. These two areas are discussed together due their symbiotic nature. Section 3 discusses distributed database research. Critical issues in the design, implementation and evaluation of distributed software systems are itemized in Section 4.

Given that distributed systems have been under investigation for a number of years now, a survey of issues in this area could have a very wide scope. We focus only on the issues connected with those software aspects mentioned above. For instance, our presentation here does not directly include such areas as the communication subnet, parallel algorithms, hardware architectures for distributed systems, and impact of other areas such as artificial intelligence on distributed systems.

2.0 DISTRIBUTED SYSTEMS SOFTWARE

Distributed systems software is usually designed and implemented in multiple levels. Each level provides the next higher level with an abstract distributed machine consisting of resources and primitives for using them. This section describes distributed systems software issues and current research for the operating system and programming language levels as well as their interaction.

2.1 Background

2.1.1 Operating Systems - Due to the wide variety of distributed systems there is a wide variety of operating systems controlling them. For purposes of discussion we simplistically divide these operating systems into three classes - network operating systems (NOS), distributed operating systems (DOS) and distributed processing operating systems (DPOS).

Consider the situation where each of the hosts of a computer network has a local operating system that is independent of the network. The sum total of all the operating system software added to each of these network hosts in order to communicate and share network resources is called a network operating system. The added software often includes modifications to the local operating system. In any case, each host can still act independently of the network and various degrees of sharing are possible. The most famous example of such a network is ARPANET and it contains several NOS's, e.g., RSEXEC and NSW [FORS78]. Such operating systems are characterized by their having been built on top of existing operating systems.

Next consider networks where there is logically only one native operating system for all the distributed components. This is called a distributed operating system. The confusing element is that this one OS may be implemented in a variety of ways. The most common implementation is to replicate the entire OS at each host of the distributed system. One might be tempted to then consider this replicated piece as a local OS. In fact it is not because resources are allocated in a more global fashion, there is no exclusive, local administrative control, and, in general, there is no dichotomy from the user's point of view between being on the network or not. If we draw an analogy to the NOS we see that the local replicated piece of the DOS is similar to a piece of an NOS with a null local OS. On the other hand, there is no requirement for a DOS to be implemented by replication and many implementation designs are possible. For example, see any of the DOSs in Table 1. A final point that should be made is that distributed systems with DOSs are designed and implemented with network requirements in mind from the beginning in contrast to NOSs where the network aspects are usually afterthoughts.

Finally, a distributed processing operating system is a DOS with the added requirement that the OS must be implemented with no central data or control at any level. This is such a demanding requirement that there are probably no systems that completely meet these requirements if, in fact, they can be met. If such systems were possible significant advantages are hypothesized [ENSL78, JENS78, STAN79]. If such systems are not possible, those approximating a DPOS would also provide similar advantages but to a somewhat lesser degree.

In Table 1 we attempt to categorize some of the current distributed computing OS research. We apolgize for those systems omitted and note that the classification is somewhat subjective. Strictly speaking, all the OS's in the DPOS category are probably DOSs. However, those DOSs that attempt to treat the special requirements of DPOSs are placed in the DPOS category. For additional information we list four distributed file systems.

TABLE 1: CATEGORIZATION OF OPERATING SYSTEMS FOR DISTRIBUTED COMPUTING

NOS	DOS	DPOS
NSW [FORS78]	Accent [RASH81]	ADCOS [STAN81]
RSEXEC [FORS78]	Apollo [APOL81]	Archons
XNOS [KIMB78]	DCS [FARB73]	CHORUS [GUIL82]
	Domain	Fully DP
	Structure [CASE77]	System [ENSL80]
	Eden [LAZO81]	HXDP [JENS78]
	RIG [BALL76,LANT80]	Medusa [OUST80]
	ROSCOE (Arachne) [SOLO79]	Micros [WITT80]
		StarOS [JONE79]
	TRIX [WARD80]	
	UNIX (5)	
	WEB [HAMI78]	

FILE SERVERS

Cambridge FS [DIO80]
DFS [STUR80]
Felix FS [FRID81]
Locus [POPE81]
Violet [GIFF79b]

The (5) after UNIX indicates that there are at least _ extensions to UNIX for distributed systems including Locus, [RASH80], and those extensions done at Bell Labs [LUDE81], Berkeley [ROWE82], and Purdue.

2.1.2 Programming Languages - Languages for distributed systems, in general, contain message-based mechanisms for process communication and synchronization. Though most extant distributed systems have been programmed using adhoc modifications to sequential languages, concurrent languages based on shared variables for process interaction are becoming more widely available. Illustrative languages in this category have been listed under Concurrent Programming Languages (CPL) in Table 2. Such languages provide high-level constructs for the specification of processes and the interaction between processes. Process interaction is via resource sharing controlled by mechanisms based on the monitor concept [HOAR74]. A monitor is a structured mechanism which synchronizes access to shared resources.

Although the shared variable based approach is appropriate for processes with shared memory, it is not suitable for programming systems in which physically distributed processes cooperate and interact closely in order to perform a single task. For such systems, due to the presence of an underlying communication network, a message-based interaction is the most appropriate. Languages designed to make this possible are called Message Passing Languages (MPL) and are listed in Table 2. However, in general, programs are written such that various modules in a program interact via procedure calls. In order to give users a uniform interface, irrespective of whether the program is executed on a centralized system or on a distributed system, a few languages have been designed to permit process interactions via procedure calls to remote sites. The remote procedure calls themselves would be implemented through an underlying message passing protocol. We call such languages as Remote Procedure Languages (RPL). The

possibility of interactions through message-passing and procedure calls is essential in order to configure tightly-coupled processes as a single logical module that execute on a single processor; processes within a module would interact via procedure calls whereas processes in different modules would interact via message-passing. Hence some of the proposed languages provide for interactions via both message passing and remote procedure calls. In the following table they are termed Hybrid Languages (HL). Details concerning these languages such as program structuring and communication primitives, can be found in subsequent sections.

TABLE 2: CATEGORIZATION OF PROGRAMMING LANGUAGES FOR DISTRIBUTED COMPUTING

CPL	MPL
Concurrent Pascal [BRIN75]	CSP [HOAR78]
Modula [WIRT77]	Gypsy [GOOD79]
Mesa [MITC79]	PLITS [FELD79]
CLU [LISK79]	
RPL	**HL**
DP [BRIN78]	*MOD [COOK79]
Argus[LISK82]	SR[ANDR81]
Ada [DOD80]	

2.1.3 Issues -

The following are the main issues involved in the design of distributed systems software.

- o Structuring distributed systems,
- o Addressing distributed processes and resources,
- o Communication between distributed processes,
- o Nature of communication channels,
- o Task Scheduling
- o Decentralized control,
- o Protection mechanisms,
- o Error recovery and deadlock, and
- o Distributed file systems.

For many of these issues much of the distributed systems software research has been experimental work in which actual (prototype) systems are built. However, further work needs to be done in the evaluation of these systems in terms on the problem domains they are suited for, their performance, etc. The prototype work is valuable because many individual research ideas are integrated into an actual system and it is important to know how well they coexist.

Of course there have also been investigations carried out independently of a particular system. This work is also treated in the discussions below. Overall, we feel that all the above issues are still unresolved in the context of NOS, DOS, DPOS, and programming languages for distributed systems and merit further work. In the remainder of this section we briefly treat each of the above research topics using examples from current research.

2.2 Structuring Distributed Systems

A node in a distributed system is a collection of resources and processes, and is typically an autonomous entity. Processes outside a node access the processes and resources within a node via messages whereas intra-node communication is normally through shared memory (HXDP is an exception). Definition of the structure of a node requires the specification of a name for a node, the resources and processes that constitute the node and a set of interfaces for accessing these processes and resources. The structure of a node could change with time.

Most distributed systems are structured using the so called resource-server model. This model in conjunction with object-orientation also appears to be a conceptually attractive structuring mechanism. There has been considerable work on both centralized [WULF74, WILK79, KAHN81] and decentralized [JONE78a, POPE81] object based systems in the context of operating systems and programming language levels [LISK82, GOLD81]. Conceptually, an object is a collection of information and a set of operations defined on that information. In practice, the term object is viewed simply as a collection of information with common access characteristics. In either case, objects seem to provide a convenient and natural model for reducing the complexities in distributed operating systems. For example, moving a process to another processor after it has begun execution requires careful treatment of code, data and environment information. Treating each of these pieces of information as objects (or possibly the combination of them as objects) facilitates their movement. Typically, associated with each object is a server which

controls the execution of operations on the object and services requests for operations on the objects.

Programming languages have started addressing the structuring issue, in particular, Argus [LISK82] (where the concept of a "guardian" is introduced), SR [ANDR81] (where the term "resource" is used) and *Mod [COOK79] (where "modules" are used for structuring systems). Processes and data within each guardian, resource, and module exist and run on one physical node. In Argus, a distributed program consists of a set of guardians. Each guardian encapsulates a set of resources, which are instances of abstract datatypes. Operations on these resources are controlled by the guardian which also provides schemes to handle concurrent access and system failure. These issues are discussed in detail in sections 2.4 and 2.9. In SR, a resource is defined along with a set of processes, called operations, to access the resource. In *Mod, a module is defined by the data structure definitions, procedures, processes and an external interface and is thus similar to a resource in SR. In all three cases, operations on the objects are executed via remote procedure calls. Using the above proposals, a distributed system can be structured as a set of objects, i.e., as guardians, resources or modules and processes that access the objects.

Only Argus addresses the issue of failures of nodes by invoking concepts such as locking, atomicity, and checkpointing, concepts traditionally applied in databases. The problem of structuring distributed systems is complicated by the presence of truly distributed resources such as distributed databases wherein resources need to be partitioned and replicated for achieving reliability and availability (see section 3). Node structuring schemes adopted in the above

languages do not appear to be appropriate for distributed database applications.

Another structuring scheme is based on the physical structure of the model associated with the problem. For example, a problem which requires solving a set of independent subproblems can be structuted as a system which first seeks the solutions for the subproblems in parallel. Chang [CHNG82] has also described some algorithms for detecting properties of general graphs by traversing paths in a graph in parallel. One of the advantages of this approach is that this utilizes the concurrency in the problem in a fairly natural manner. Programs written in functional languages and implemented on data-flow architectures are amenable to this form of structuring [COMP82].

2.3 Addressing Distributed Processes And Resources

The addressing issue [SALT78, SCHO78, DAVI81, BIRR81] in distributed systems is complicated because "names" are used in the context of so many different functions and levels. For example, names are used for referencing, locating, scheduling, allocating and deallocating, error control, synchronization, sharing, and in hierarchies of names - to name a few. Names actually identify a resource in a logical way, then are mapped to an address which in distributed systems is then mapped to a route. Names used at different levels referring to the same object must also be bound together. Allowing relocation and sharing of objects causes additional problems. Ideally, names should be bound to addresses and routes dynamically for

increased flexibility, but this causes a potentially large and excessive run time penalty. Broadcasting is often involved in implementing dynamic binding of names, addresses and routes. Heterogeneous distributed systems cause additional naming problems (more mappings required).

Addressing can also be categorized as implicit, explicit, path-based, or functional [VINT83].

The simplest form of connection establishment is implicit addressing, where a process can only communicate with one other process in the system. This is usually the case for processes created to perform a single service and only communicate with their parent process. Though such a model is appealing in its simplicity, it is not flexible enough by itself to allow any pair of processes to communicate, a minimal requirement of a general communication facility.

Explicit addressing is characterized by the explicit naming of the process with which communication is desired [HOAR78, BRIT80]. This addressing scheme is particularly suited for process configurations in which the output of one process serves as input to another. Explicit addressing requires a global knowledge source containing the identity of each process in the system. Some systems have predefined names for processes providing system services and a user accessible table of user process ID's. A communication mechanism dependent on explicit addressing is an adequate basis for a complete communication system, as evidenced by its use in the Thoth system [CHER79]. However, explicit addressing by itself is not flexible enough to effectively handle such common circumstances as process migration in distributed systems and

multiple processes providing a single service.

Path-based addressing associates global names with message receptacles, or 'mailboxes'. A process can declare a mailbox into which messages can be received and can specify a destination mailbox when sending a message. Such a scheme is used in [GOOD79] wherein the mailbox is global to the processes that use it. Recently, use of a distributed but globally accessible memory for process communication is described in [GELE82].

Functional addressing establishes a connection based on the need to serve or request a service. In this case the communication path is itself a named entity of the system. For the user of a path, the identity of the process or processes on the other end of the path is insignificant. What is significant is that they are providing a service or that they are requesting that a service be provided. This scheme is flexible because an individual process is not necessarily associated with a communication path, and paths themselves may be passed within messages. This kind of addressing was introduced in [BALZ71] and expanded in [WALD72]. Functional addressing is the underlying concept used in many current languages, most notably in the following: PLITS [FELD79] and *Mod [COOK79] where "port" denotes a path, Distributed Processes [BRIN78] and Ada [ADA80] where "entry" denotes a path, and SR [ANDR81] where "operation" denotes a path. It is also used in the following message-based operating systems: Accent [RASH81] where 'port' denotes a path, and DEMOS [BASK77], where 'link' denotes a path.

Addressing in most programming languages is static in that the processes accessible to each process is fixed at compile time. However, some languages, for example, Gypsy [GOOD79] and PLITS [FELD79], do support dynamic addressing whereby it is possible to interact with a changing environment.

It should be clear that the various forms of addressing described above need different kinds of operating system support. Implicit addressing is straightforward since a process is connected to only one other process. Translation from logical name to physical address is necessary for explicit addressing. In path-based addressing, access to mailboxes has to be mutually exclusive. Even here, mapping from logical mailbox names to physical mailbox addresses has to be performed. Functional addressing requires the most operating system support. This is due essentially to the dynamic nature of path connections between servers and the served. The first element of address mapping is the determination of the process that provides the needed service followed by communication with the process along the path connected with the process. Operating System support for functional addressing is described in detail in [STEM82].

2.4 Communication Between Distributed Processes

There are three aspects of inter process communication (IPC): communication primitives used for message-passing, techniques used for bufferring messages, and the structure of messages [VINT83]. (In this section and those that follow, the term "port" refers to a communication link between two processes.)

We start our discussion of the first of the above aspects, namely communication primitives, with procedure calls. This primitive is commonly used for programming resource-server interactions and hence languages provide higher-level language support for this type of interaction. A procedure call can be simulated in a message oriented approach to process communication with the reply-send primitive (also referred to as the remote invocation send [LISK79]). (Consult [NELS81] and [STAN82a] for an extended comparison of message-based and remote procedure call approaches to communication.) The remote procedure call can be implemented either by having a single cyclic process, as in *Mod [COOK79] and SR [ANDR81] for each type of call, by creating a new process for each execution, as in Distributed Processes [BRIN78] and Argus [LISK82], or by programming the server process with separate receives for distinct calls, as in Ada [ADA80]. With the synchronized send, the sender is blocked until the message is received by the destination process [HOAR78].

The reply-send and synchronized send primitives require no message buffering or queueing if only a single process has send access to the path.

The no-wait send (or asynchronous send as discussed in [GENT81]) maximizes concurrency between communicating processes. In this model, the sender resumes execution as soon as the message is composed and buffered within the communication facility. The no-wait send is the most flexible of the three send primitives, though it introduces extensive implementation problems. The need for buffering introduces problems of flow and congestion control, addressed later in this subsection.

There are two forms of receives: unconditional and conditional. The unconditional receive, or blocked receive, blocks the receiver until a message is queued on the selected port. The unconditional receive sacrifices process concurrency by blocking the receiver when no messages are queued. It has two variations. Frequently it is desirable to block on a set of paths. The first message received on any of the specified set of paths is returned to the receiver. This primitive, known as restricted unconditional receive is useful for a serving process awaiting messages from multiple sources. A second, more important variation is the blind unconditional receive. This form of unconditional receive blocks on all paths to which the requesting process has receive access [STEM82].

The conditional receive (or selective receive as discussed in [RA080]) introduces a polling capability, allowing the receiving process control over the degree of concurrency desired. The conditional receive polls a (set of) port(s) for a queued message. If a message is present it is returned, otherwise control is returned with a flag indicating that no message is available. The conditional receive can be generalized to check for a more complex condition than the presence or absence of a message [ANDR82].

The second aspect of communication between processes involves implementing the buffering of messages. Regardless of the buffering technique used, buffer maintenance introduces the problem of resource allocation. There are three basic techniques, port-local allocation, process-local allocation, and system-global allocation, and a hybrid technique called port-global allocation.

Port-local allocation, assigns a fixed buffer space to each port [RASH81]. When the buffer space is filled and a process attempts to send another message to the port, there are three alternatives. First, if a flow control option is included in the send primitive, the port can dynamically expand its size to allow an additional message. Second, the sending process can remain blocked until space is available on the port for an additional message. Third, an error flag can be returned to the sending process indicating a full port.

The process-local allocation technique associates the buffer space with the process rather than the port [KNOT75]. This restricts the total number of outstanding messages for a process.

A third resource allocation alternative is to maintain a global buffer pool (examples include Roscoe [SOLO79] and UNIX [RITC74]). This technique, termed system-global allocation, introduces a bottleneck in the system. A viable solution to the two problems mentioned above is to combine the port-local and system-global techniques. The port-global hybrid associates a fixed buffer space to each port while reserving buffer space with the system.

The third aspect of communication is the structure of messages. An important issue in message structure is the typing of data within messages. Some systems are purporting the benefits of strongly typed data within messages (Eden [LAZO81], CLU [LISK79], and Accent [RASH81]). Strong typing of data restricts the contents of messages being passed and thus provides reliability in general, but in a distributed system consisting of heterogenous nodes, it is especially important in ensuring proper conversion of messages across nodes. However heterogeneous

computer networks [ANDE71, LEVI77, BACH79] cause problems because of the different internal formatting schemes that exist. One proposed solution to the data incompatibility problem is to define a canonical representation for each type that can be used in messages. Each different implementation of the type defines a translation function between its representation and the canonical representation [HERL82]. Further complications arise when there are precision loss and data type incompatibility, internetwork mappings (gateways), including naming [SCHO78] and routing.

Another issue concerning message structure is the length of the messages. The Roscoe [SOLC79], StarOS [JONE79], and Thoth [CHER79] systems have short, fixed length messages. This is in part due to the belief that most messages in such tightly coupled systems are short control messages. Special, synchronous communication paths are provided for large data transfer, such as reading and writing files. Medusa [OUST80], Accent, and CLU [LISK79] provide for variable length messages, and, in general, most protocols for loosely coupled systems support variable length message transfers. Variable length messages are obviously more difficult to implement due to the buffering problems they induce.

2.5 Nature Of Communication Channels

The primary considerations in the design of communication channels in a distributed system are directionality, ownership, frequency of use, transference rights, and connection structure [VINT83].

Directionality concerns whether or not a single process has both send and receive access to a single communication path. If so, the path is bidirectional (or duplex, as used in TRIX [WARD80]). Most systems provide only unidirectional (or simplex) paths, in which a process may send or receive messages over the path, but not both. In systems providing only unidirectional paths, bidirectional communication can be simulated via a pair of paths.

Ownership deals with the capability to destroy the path and terminate communication without consent of all processes with access to the path. Ownership can be associated with the directionality of communication, as in the Accent system where the allocator of a path automatically has ownership and receive access to the path.

Frequency of use deals with the limitations on the number of times a path can be used. The DEMOS [BASK77] and Roscoe [SOLO79] systems have a 'reply' path, created for the sole purpose of returning a single message. Such a path is automatically destroyed after it is used once.

Transference rights concern the ability to duplicate a path, or pass access and/or ownership rights of an established path to another process. For example, a process with ownership rights to a path can send that privilege to a receiving process, allowing it to destroy the path or change the path's characteristics. Transferring access rights can, but does not necessarily, imply loss of the transferred right.

Connection Structure of a path is the nature of process connections established by the path. Connection structure can take four general forms: one-to-one, one-to-many, many-to-one, and many-to-many. Communication paths that are not one-to-one can often be functionally

equivalent to a set of one-to-one paths, an important issue in examining connection complex connection structures. The Process Control Language (PCL) [LESS80] provides a complete specification of a wide variety of topological structures of unidirectional communication paths. The terms used below are introduced in the PCL description.

The simple path is provided for a one-to-one connection. A single queue is maintained to hold all messages. The broadcast and multiple read connections are one-to-many connections, associating a set of receivers with each sender. Every message sent on a broadcast communication path is received by every receiver and hence is suited for system wide dissemination of information; any message sent on a multiple read path is received by the first process requesting it and hence is suitable in situations where multiple servers exist for a single type of service and the service is performed equally well by any of the servers. The broadcast path can be easily simulated by a set of simple paths from the sending process to the set of receivers assuming all the receivers are known. A multiple-read path cannot be simulated with one-to-one paths without additional management control within the processes performing the service.

Many-to-one connections are provided in the multiple-write and concentration communication paths. A multiple-write mapping is the converse of the multiple-read mapping: every message sent by any sender is received by the receiver. This connection which is useful when multiple users are served by a single server and can be simulated by forming a set of simple paths from the sender processes to the receiver. The concentration path can be used for synchronization of the set of senders: every message received is the concatenation of the set of

messages from a single send by each sender. A message cannot be received until every sender has transmitted a message. Thus a concentration path can be used to connect processes, that are solving a set of subproblems in parallel, with the process awaiting the result of the subproblems.

Many-to-many paths are combinations of one-to-many and many-to-one paths. The multiple-write/broadcast transmits every sent message to each receiver. A message transmitted on a multiple-write/multiple-read path by any sender is only received by the first receiver requesting it. A concentration/broadcast message is the concatenated messages from each sender and is sent to every receiver. A concentration/multiple-read message is the concatenated messages from each sender and is received by the first receiver requesting it.

The connection structure of a path is closely related to the transference and access rights of processes to the path, and may change over the lifetime of the path.

Current programming languages do not appear to have the facilities to specify the above restrictions on communication paths. However, a need for such facilities is apparent, especially for the designer of a protected system [STEM82] wherein protection of resources and processes is achieved by restricting the use of communication paths.

2.6 Task Scheduling

Task scheduling research for distributed systems has been approached in a variety of ways including task assignment, job scheduling, and clustering. While ideas from these various approaches overlap, most of the research on scheduling for distributed systems can be considered "task assignment" research and can be loosely classified as either graph theoretic [STON77, STON78a, STON78b, BOKH79, CHOW82], based on mathematical programming [CHU69, CHU80, MA82], or heuristic [GONZ77, ELDE80, CHOW82, EFE82]. By task assignment is meant that a task is considered to be composed of multiple modules and the goal is to find an optimal (or in some cases a suboptimal) assignment policy for the modules of an individual task. Typical assumptions found in "task assignment" work are: processing costs are known for each module of the task, the interprocess communication (IPC) costs between every pair of modules is known, IPC cost is considered negligible for modules on the same host, and reassignment does not occur.

An example of task assignment research where reassignment does occur can be found in the "domain structure" operating system [CASE77]. The scheduling algorithm in this system directly addresses the scheduling and dynamic movement of individual tasks. It does this by describing the structure of a task that is necessary to facilitate movement of tasks that are in execution. However, the scheduling algorithms are completely heuristic and contain weighting factors which are left unspecified, presumably to be tuned as system parameters. Interaction between multiple tasks in the system and the necessary scheduling decisions to take these interactions into account are not discussed.

Task assignment research for extremely large distributed systems has also received some attention. Micros [WITT80], for one, has a unique scheduling algorithm called Wave scheduling. The Wave scheduling algorithm co-schedules (assigns) groups of related tasks onto available network nodes. The scheduling managers themselves are distributed over a logical control oligarchy and send waves of requests towards leaves of the control oligarchy attempting to find enough free processors. In fact more processors than required are requested because some parts of the control oligarchy may not be able to supply the necessary processors. This is a form of probabilistic scheduling.

The second form of scheduling research on distributed systems, referred to as job scheduling, can be thought of as assignment of entire jobs to processors where jobs are independent of each other. Approaches to job scheduling have included bidding [FARB73, SMIT80], queueing theoretic approaches [CHOW79, KLEI81, AGRA82], the use of estimation theory [BRYA81], and statistical decision theory [STAN83a].

As an example of job scheduling, bidding schemes attempt to match specific tasks to processors based on the current ability of the processors to perform this work. These schemes are suboptimal, but are more extensible and adaptable than many of the other approaches. However, the cost of making and aquiring bids may become excessive, and the factors to use in making the bids have not been extensively studied.

The Distributed Computer System (DCS) [FARB73] was the first system to perform a form of dynamic load balancing using a bidding technique. This system used percentage of available memory as a load indicator. Incoming jobs are then routed to processors with the most memory

available. Information concerning current system status is gathered by means of a broadcast message. However, no analysis, comparisons or measurements were performed to determine the effectiveness of this scheme.

Distributed scheduling research known as clustering refers to scheduling highly communicating tasks on the same processor. "Highly communicating" implies that there is a large amount of data transfer between tasks of the cluster, or a high frequency of data transfer between the tasks of the cluster, or both. Two operating systems StarOS StarOS [JONE79] and Medusa [OUST80], both implemented on the Cm multi-microprocessor [SWAN77], deal with clustering.

StarOS is a message-based, object-oriented multiprocessor operating system. One main idea of StarOS is the task force, a large collection of concurrently executing processes that cooperate to accomplish a single task. The structure and composition of a task force varies dynamically and it is the unit for which major resource scheduling decisions are made. Even though StarOS is designed for a multi-microprocessor, we believe that the task force concept could also be used for more loosely coupled distributed systems. The scheduling function itself is divided between processes which are called schedulers, and a low level mechanism called the multiplexor. The schedulers decide which environments are to be loaded and the multiplexor performs the actual loading. As far as we know only very simple scheduling algorithms were implemented.

Medusa [OUST80] is an attempt to capitalize on the architectural features of Cm*. Medusa is implemented as a set of utilities (OS functions), each utility being a task force (task force has the same meaning as described above for StarOS). Each utility contains many concurrent, cooperating activities. Load balancing is done by automatically creating new activities within a task force to handle increased load and automatically deleting activities when the load is reduced, by coscheduling (an attempt to have different activities of the utility executing simultaneously on different hosts), and by pause time (a short time in which the context of a process remains loaded after an interrupt to determine if it will be reactivated). The pause time concept is supposed to reduce context swaps.

As a final note on scheduling, we note that there is a similarity between some of the above described scheduling research and research in routing algorithms. For examples see [MCQU74, GALL77, SEGA77].

2.7 Decentralized Control

Various forms of decentralized control of resources are appropriate to our current discussion: decentralized control that arises in distributed databases, decentralized control that arises for stochastic replicated functions, such as routing and scheduling, and decomposition, which is a form of decentralized control that potentially arises in implementing many functions in a distributed system.

In distributed databases, concurrency control algorithms are sometimes implemented as decentralized control algorithms. The decentralized concurrency control algorithms must maintain database

integrity. This constraint reduces potential parallelism but is necessary. Solutions to the type of decentralized control are known (see section 3).

By replicated functions we mean that the decentralized controllers implementing a function are involved in the entire problem, not just a subset of it. By stochastic is meant that there is no data integrity constraint and information is noisy and out of date. In contrast to the decentralized control problem for databases, solutions (in the most general form) to decentralized control of stochastic replicated functions are not known, e.g., Team theory [HO80] and various forms of control theory all fall short in dealing with this problem [STAN82b]. Only preliminary work has been done in the area of decentralized control for stochastic replicated functions [LELA80, STAN82c, STAN83b]. Possible approaches might include the use of random graphs, decision theory [RAIF61, HALT71, WINK72, LIND81, STAN83a], and stochastic learning automata [NARE74, GLOR80]. The complicating issues for stochastic replicated functions include the need for very low overhead solutions, the operation in the presence of noisy and delayed information, the high degree of interaction between cooperating controllers, and the fact that decisions of each controller affect the others.

Although research has been active for all three types of decentralized control mentioned above, the majority of the work is based on extensions to centralized solutions where the entire state is known and can be more accurately described as decomposition techniques, rather than decentralized control [CHAN69, FU70, JARV75, CHON75, AOKI78, HO80]. In such work large scale problems are partitioned into smaller problems,

each smaller problem being solved, for example, by mathematical programming techniques, and the separate solutions being combined via interaction variables. The interaction variables normally model very limited cooperation. See [LARS79] for an excellent summary of these types of decentralized control (decomposition). Yet other surveys have appeared including [SAND78] that note the unclear meaning of optimality for decentralized control and hypothesize the need for a completely different approach. One approach for decentralized control is based on the concept of a "domule". A domule is the combination of a decision agent (controller), its local subsystem, and interaction relations between agents [TENN81a,b]. Using this concept, interesting heuristics are proposed for decentralized control but these are largely based on decomposition. Many applications in distributed software may be adequately addressed by known decomposition techniques. However, to apply these techniques many of the complicating issues listed above for stochastic replicated functions would have to not exist or be minimal. In summary, decentralized control remains a rather perplexing issue.

2.8 Protection Mechanisms

Protection has been one of the main concerns of operating systems from their conception [GRAH68, SALT75]. The notion of capabilities, an often occurring notion in the area of protection, was first introduced into an operating system by Dennis and Van Horn [DENN66]. Systems employing a capability mechanism view all data as strongly-typed objects which are distinguished from one another by unique identifications. A capability consists of an object identifier and a set of access rights, which allow the manipulation of the object with a subset of the

operations defined by the object's type. Capabilities were later incorporated into the CAL [LAMP76] and Hydra [WULF74] systems, and implemented in the Plessey System 250 [COSS72]. The Multics system implemented a complete capability-based scheme with hierarchical control of authorization [SALT75]. It used segmentation as the memory addressing scheme wherein protection is achieved by associating with a computation concentric rings of decreasing access privileges. [PASH82] compares some of these systems from an architectural point of view.

Capability-based protection mechanisms are difficult to implement and entail a certain amount of overhead in that each access needs to be checked. Once a process obtains a capability it is essential that it not be able to modify it. Capabilities may be stored as C-lists, as with the Intel 432 [KAHN81], or as tagged memory, as with the IBM System/38 [BERS80]. The C-list scheme stores all capabilities in separate capability segments. A major difficulty in the use of capabilities is the separation of data and capabilities. Tagged memory requires each unit of data, 32 bits in the case of the System/38, to be tagged to indicate whether it is a capability or not. Thus tagging presents a large overhead for memory. See [LEVY81] for a complete description of capability-based architectures.

Object-oriented systems such as CAL [LAMP76], CAP [NEED77], Hydra [WULF74] and Intel 432 [KAHN81] associate groups of objects with modules. A module consists of a set of processes and local data in addition to shared objects. A process within a module has the right to distribute access rights to processes outside the module. Programming language constructs for expressing controlled access to shared objects

is the subject of [JONE78b]. These can be used in conjunction with languages designed using the concept of abstract data types such as CLU [LISK79].

2.9 Error Recovery

Exceptions in distributed systems can occur due to data transmission errors and process control errors. Data transmission errors, including lost messages, the receipt of garbled messages, duplicates, and misdirected messages should be transparent at the process level. It is the responsibility of the communication facility and its underlying protocols to ensure reliable, error free data transmission across communication paths between processes. This requirement results in all remote procedure calls having an exactly once semantics whereby a call terminates after the called procedure has been executed exactly once in spite of system failures.

Process control errors take the form of processes involved in a deadlock or a livelock, and destroyed processes (including node failure). The deadlock problem has been studied in various contexts. See [ISLO80] for an excellent survey of deadlock in centralized systems. Additional work has been published for distributed systems but it is largely concerned with resources in distributed databases (see section 3).

Little has been written about deadlock in the context of NOS, DOS and DPOSs. One exception is the Medusa system where deadlock avoidance is used. Here functions provided by the utilities (OS functions) are divided into service classes such that (i) a single utility provides all

the services in each class, and (ii) there are no circularities in the dependencies between classes. Furthermore, each utility must contain separate pools of resources so that it can provide independent service to each class. These conditions avoid the deadlock problem. More work is required for all approaches to deadlock in distributed systems.

Now we examine specific cases of deadlock. The blocked receiver problem occurs when a receiver blocks on an unconditional receive and no message is ever delivered to the process. Unexpected process destruction is the most likely candidate for causing a blocked receiver problem. The IPC facility must take responsibility for notifying other processes attached to the destroyed process. The IPC facility should notify the receivers connected to the communication paths to which the destroyed process has send access. Processes with send access to a port connected to a destroyed receiver should be notified on the next attempt to send a message.

The deadlock problem is a direct result of blocked or destroyed processes preventing further communication. If blocked receivers timeout and senders are properly notified, deadlock cannot occur. If either of the features are not included in the communication facility, it is necessary to actively detect deadlock.

Programming language designers have only recently started paying attention to the specification of actions to be taken in the event of a failure. An often adopted solution to exception handling is a time-out, in which the processes indicate a specified time limit for waiting. A sending process specifies the time within which it expects its message to be received whereas a receiving process specifies the time within

which it expects the next message to arrive. When the time limit is exceeded, control returns to the process concerned with a message indicating the time expiration [ADA80, LISK82]. In PLITS [FELD78], the notion of transaction keys can be used to notify processes about errors. For instance, a process can be programmed to wait for a message by transaction key only, without specifying the source of the message. Thus, failure of a sending process can be intimated to the receiving process by using the appropriate transaction key.

Also, in Argus [LISK82] the notion of atomic action is used to cope with errors. An atomic action appears to a user to have been executed in exclusion and thus gives users the facility to program remote procedure calls with the exactly once semantics. Their implementation requires facilities for locking and unlocking data items such as those available in databases. Recovery and roll-back to some previous system state may be another option. In [RUSS80] a set of primitives are proposed for state restoration in distributed systems.

An entirely different approach, still in its infancy, to dealing with errors in distributed systems is to construct provably-correct systems from the start. Gypsy [GOOD79] facilitates the development of formally correct programs along with exception handling mechanisms. Proofs make use of the history of message passing in the system. Several proof methods have been proposed for CSP programs [APT80, CHAN81]. Proofs for the absence of deadlock in distributed systems can be found in [CHAN79]. Proofs of systems structured using the resource-server model are presented in [RAMA82]. The issue of specifying and proving network protocols has also been receiving attention. Some of the approaches are based on history variables

associated with message passing [GOOD79] or on state machine models [BOCH78]. Recently techniques based on temporal logic [MELL81] have also been investigated. These attempts have to be further developed before they can become practicable for constructing correct distributed systems.

2.10 Distributed File Systems

Research in distributed file systems can be considered as providing an important but only incremental step in establishing integrated distributed computing systems. Three representative distributed file systems (see Table 1) are now briefly described in order to illustrate what facilities current distributed file systems provide. This section then leads into a discussion of distributed database issues which subsume the issues involved here.

The Xerox Distributed File System (DFS) [STUR80] is used as a basis for database research. Overall, DFS gives the illusion of a single, logical file system, and it runs on a local network. To support this illusion there exists a file locating facility that makes file location transparent to the user. The DFS is based on a client-server model where multiple servers (one being designated the primary) may cooperate to service a single transaction in an atomic fashion. Multiple files, located across the network, may be involved in a single atomic transaction. File replication is not supported. Users access the DFS through programs called clients that run on the user sites. DFS uses a locking mechanism between transactions that supports client caches for

increased efficiency. Finally, facilities exist to deal with server crashes and aborted transactions.

The Felix file server [FRID81] is designed to support a variety of file systems, virtual memory and database applications by providing a simple interface. Felix is the base upon which a higher level file system is to be built. Felix is the only storage component in the system and runs on a local area network. An atomic transaction in Felix may involve single files or a set of files. A highly flexible mechanism for file sharing is supported by using six access modes specified when the file is opened (read copy, write copy, read original, write original, read exclusive and write exclusive). The level of locking granularity is the block level. Access control is based on capabilities and no file replication is supported. Felix is also based on the client - server model.

LOCUS [POPE81] is a distributed file system that is application code compatible with UNIX [RITC74]. In contrast to the above two systems LOCUS does support file replication. A centralized synchronization mechanism is used to maintain mutual consistency among replicated files as well as to synchronize multiple accesses to shared files. LOCUS continues to operate even though there is partial system failure or network partitioning. A concept based on version vectors is used to resolve conflicts at system recovery time. LOCUS is not designed as a client - server model but as a file system integrated with the rest of the operating system.

Many of the open issues for distributed file systems are the same as for distributed databases (see Section 3). These include how to maintain the atomic property in the presence of crashes and other failures, support of multiple copies, concurrency control and deadlock resolution. Other issues include directory assignment, replication and partitioning, division of the responsibility between file servers and clients, and the relationship of the distributed file system to the operating system.

3.0 DISTRIBUTED DATABASE MANAGEMENT

In this section we give a brief survey of the current research associated with the development of general purpose distributed database management systems (DDBMS's). Rothnie and Goodman [ROTH77] state that "distributed database mangement is an attractive approach ... because it permits the database system to act conceptually as a centralized system, while physically mirroring the geographic distribution of organizations..." Some of the potential advantages of distributed database management systems are: easy access to geographically distributed but logically integrated data from a single site, increased reliability and availability, faster data access, and incremental system growth.

The following are the main issues involved in distributed database management.

- Transaction model,

- o Synchronization (concurrency control),
- o Deadlock resolution,
- o Recovery (Failure handling),
- o Data Models,
- o Data Replication and Location Transparency,
- o Nested Transactions,
- o Distributed directory/dictionary management,
- o Typical Applications, and
- o Prototypes and testbeds.

3.1 Transaction Model

The transaction concept has emerged as an abstraction which allows programmers to group a sequence of actions into a logical execution unit. If executed atomically, a transaction transforms a current consistent state of the database into a new consistent state [ESWA76]. The virtues and limitations of the concept are described in [GRAY79]. It is the job of the transaction processing component of a DDBMS to preserve atomicity of transactions. In order to do this, protocols for resolving data access conflicts between transactions (concurrency control protocols) and protocols for recovering a consistent state of the database in spite of user errors, application errors, or partial system failure (processes, nodes, links, etc.) are necessary [KOHL81].

There is general agreement that the logical structure (system architecture) of a distributed database management system can be described by Figure 1. The four basic components are transactions, transaction managers (TM's), data managers (DM's), and data. Each user

transaction is controlled by and interacts with the data management system through a single transaction manager (TM). The TM's may simultaneously control multiple independent user transactions. The TM in charge of a transaction forwards database access and update requests to the data manager (DM) local to the data. The local DM's are responsible for managing their own stored databases and for completing local access and update commands received from TM's on behalf of transactions. The level of the access and update commands passed to the DM's depend on the implementation. They may be low level page read and write requests or high level query and update operations requiring extensive computation as well as file input and output. A more detailed description of this logical model and how it is used as a basis for concurrency control theory can be found in [BERN81a].

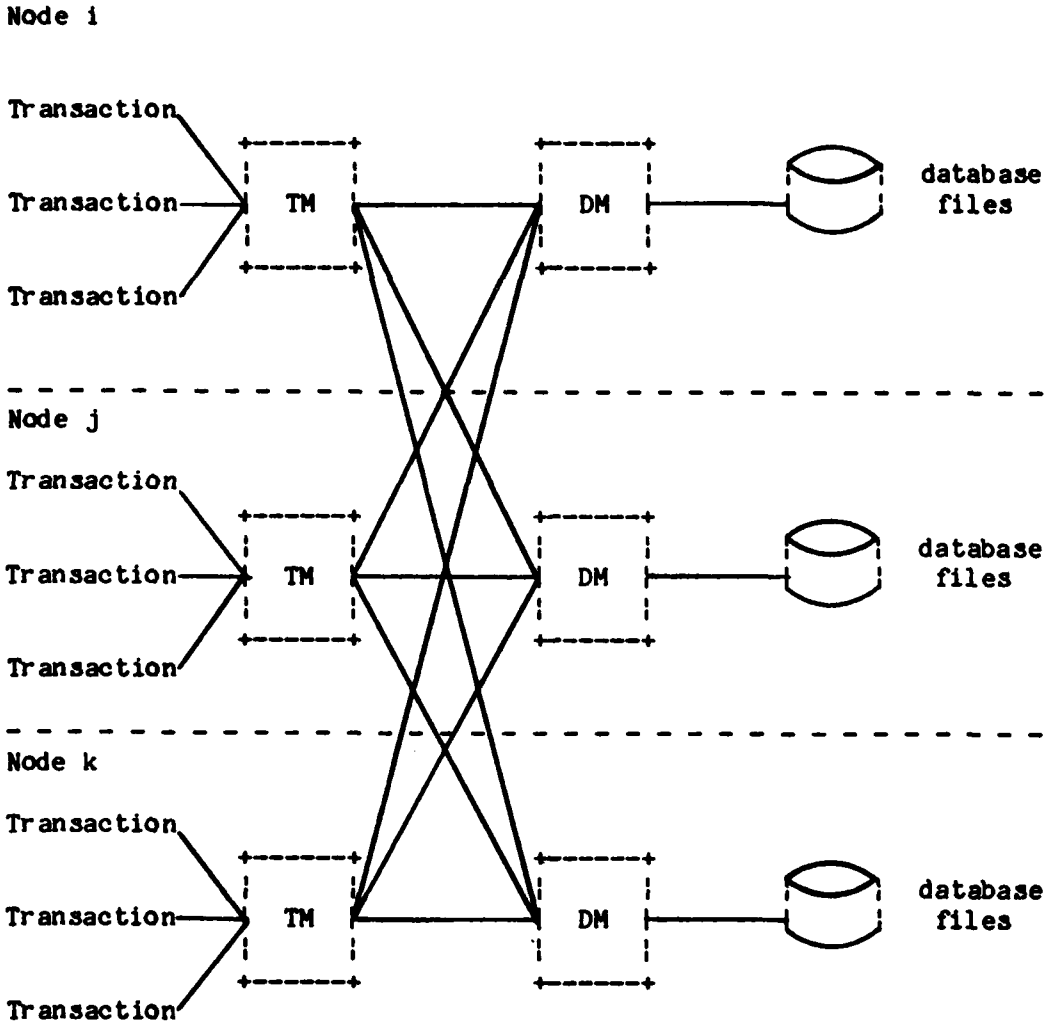


Figure 1. DDBMS System Structure.

3.2 Synchronization (Concurrency Control)

Bernstein and Goodman [BERN81a, BERN82a, BERN82b] have developed a unified framework, called serializability theory, for analyzing the correctness of concurrency control algorithms and have shown that most of the commonly known methods can be understood using a few basic concepts: read-set and write-set, schedule, serial schedule, legal schedule, conflicts, dependence relation, read-write conflict, write-write conflict, serializable schedule, and equivalence of

schedules. A schedule for a group of transactions with the property that all actions of any transaction comes before or after any actions of other transactions is a serial schedule. This means that transactions are executed one at a time in some order. Serializability theory assumes that a serial execution is always correct. A schedule which is equivalent in its effect on the database to a serial schedule is said to be serializable. The serializability theorem [BERN81a, BERN82a] states the conditions under which a schedule is serializable. See the listed references for theorems and definitions of other terms.

The theory partitions the synchronization problem into two independent sub-problems: read-write and write-write synchronization. Each concurrency control scheme is a combination of policies to solve the sub-problems and each policy can be implemented using combinations of mechanisms. The three major classes of concurrency control policies are: locking (two phase locking), timestamp ordering, and validation (also called the optimistic approach [KUNG81, SCHL81, BHAR82]). There are also a variety of algorithms to support each of these policies. Several hundred variations have been identified for the distributed environment [BERN81a, BERN82a] covering the entire spectrum from centralized to decentralized control. We will make no attempt to list the very large number of research papers which have been dedicated to the study of this topic.

New methods for concurrency control are expected to be combinations and minor variations of known methods. However, even though many variations are known, little is known about how the choice of a scheme will affect system performance. Some simulation and queueing studies [KAMO82] have compared performance, but the results are still

inconclusive. In fact, since the models used in the studies have not been validated, their results should not be accepted without close scrutiny. Several experimental studies involving testbed systems are either underway or planned. The results from these studies should be helpful in the comparative evaluation, but it is unlikely that one approach will always be the "best" due to the wide variety of applications and system structures. These studies should provide enough evidence to determine if the choice of concurrency control algorithm has a primary or secondary impact on performance compared with other factors [BALT82].

The correctness of a concurrency control algorithm is judged by the serializability of all the schedules allowed in the scheme. That is, if all the legal schedules of a scheme are serializable, then the scheme is correct with respect to the serailizability requirement.

Serializability represents the strongest degree of consistency. Some applications may require only lower degrees of consistency by allowing, for example, reads from multiple objects without concern for time consistency between the values read. Gray et al [GRAY76] discuss four degrees of consistency, degree 0 through 3, where lower degrees require fewer locks to be set or locks to be held for a shorter period of time. A smaller degree of consistency may increase performance but has the disadvantage of reducing the effectiveness of system recovery after failure. Gray et al suggest that the performance penalty of degree 3 consistency is small and consequently lower degrees of consistency are a bad idea.

3.3 Deadlock Resolution

Within database systems, deadlock is a circular wait-for condition which may arise when a locking scheme is used for concurrency control. It may occur among a set of transactions as a result of the decision to wait for a lock because of lock mode incompatibility between the granted lock and the request. The circular wait-for condition prevents any transaction from proceeding unless one or more of them are chosen as victims to be aborted. Most of concurrency control schemes using locking with delay incorporate a deadlock resolution mechanism, but a deadlock-free locking policy can be designed using deadlock avoidance or prevention mechanisms as well.

We classify the mechanisms into three categories. More discussion of these approaches can be found in [ROSE78, MENA79, RYPK79, GLIG80, CHAN82, KORT82, OBER82].

1. Timeout. One of the simplest ways to resolve deadlock is to specify a maximum wait time, and roll back the transaction which is waiting if the time expires before the request is granted.

2. Deadlock Prevention and Avoidance. Deadlock can be prevented by requiring each transaction to preclaim all the objects to be accessed before execution. In this case, if a lock request cannot be granted, all locks preclaimed are aborted.

   As another more interesting example of a prevention mechanism, we discuss briefly two protocols proposed by Rosenkrantz and Stearns [ROSE78] which use timestamps to define a priority. That is, an older transaction (one with a smaller timestamp) has higher priority

than a younger transaction. The basic philosophy is that an older transaction is favored when conflicting with a younger transaction on the assumption that it may have already used more resources and thus more cost will be paid for rollback and restart.

- o Wait/Die Protocol : If the requester (of a lock) is older (smaller timestamp), then it is allowed to wait; otherwise, it dies (die means rollback and restart).

- o Wound/Wait Protocol : If the requester is older, then the requester wounds the conflicting younger transaction, otherwise, the requester waits. "Wound" is an activity in which the requester sends messages to all the nodes the younger transaction has visited saying that it is wounded and the scheduler will abort a wounded transaction if the transaction has not initiated termination. If the younger transaction is terminating, i.e. in two-phase commit stage, then the wound message is ignored to save some unnecessary rollback/restart.

See [ROSE78, KOHL81] for more details.

3. Deadlock Detection. The detection mechanisms are mostly based on finding cycles in a transaction wait-for graph (TWFG). The nodes of the graph represent the waiting transactions, and the directed edges indicate for which transactions a given transaction is waiting. When cycles are detected, they are broken by choosing victims to be rolled back and restarted. This mechanism seems to have greater popularity than other mechanisms. However, in distributed systems a global transaction wait-for graph (GTWFG) must be constructed to detect deadlock invoving two or more nodes. The GTWFG is maintained

by exchanging messages among the nodes, causing additional communication overhead and delay. When a detection mechanism is chosen, some other decisions have to be made by the designer:

- o Detector organization: centralized detector, distributed detectors, or something in between, e.g., hierarchically organized detectors with a detector for a cluster of nodes and a higher-level detector for a cluster of detectors.

- o Detection initiation: is detection made periodically using a predefined time parameter, is it initiated every time a transaction has to wait, or when any suspended transaction has waited for more than a predefined time period.

- o Failure resiliency: establish backup for a centralized detector and activate when centralized detector fails.

The importance of an efficient detection algorithm for performance and which is the best approach are issues which have not been adequately resolved. It clearly depends on the average frequency of deadlock in the database application environment. A study by Gray [GRAY81] showed that the frequency of deadlock goes up with the square of multiprogramming level and the fourth power of the transaction size.

3.4 Recovery (Failure Handling)

It is the responsibility of the DDBMS to insure that transactions are atomic in spite of user errors, application errors, or partial system failures. This means that if a failure occurs to prevent the

successful completion of a transaction, all database entities that the transaction modified must be restored to their state prior to the transaction. The same mechanism is used by the concurrency control scheme to roll back one or more transactions to resolve deadlock. This rollback is normally achieved by using a "shadow paging" mechanism or a "write ahead log" [TRAI82b]. Operating system support is needed to make these mechanisms more efficient.

Protocols for preserving transaction atomicity are called commit protocols [SKEE81]. They are used to insure the completion (commit) or rollback (abort) of the actions of a transaction at all sites. When multiple sites are involved, the question of site autonomy arises [LIND80b]. It is well known that when the common two-phase commit protocol is used, a site loses its autonomy to independently roll back once it enters the the second phase of the two-phase protocol. Skeen calls this a blocking protocol because a site failure under certain circumstances may leave operational sites blocked waiting for the failed site to recover. He has proposed an extension, called a three-phase commit protocol, which has the property that it is nonblocking if certain assumptions about the system behavior are true. The nonblocking property is achieved by using additional messages. More research is needed to evaluate if the assumptions on system behavior are realistic and if the performance penalty of the additional messages required by the third phase are worth it.

While there has been some recent research on recovery protocols and rollback mechanisms, an adequate theory and formal model for recovery in a distributed system has not yet been formulated. More research is required to address this and related problems of failure detection,

network partitioning, and node restart and integration [VERH78, GARC82, MINO82].

3.5 Data Models

By restricting the choice of data model, there is more potential to optimize performance. The relational model has been the choice of researchers in DDBMS's. One reason is the opportunity for optimizing query processing by decomposing the query into subqueries which can be performed at remote sites [BERN81b]. The choice of a decomposition depends on the processing costs, the communication costs, and the data distribution. There continues to be some interest in developing optimal decompositions which minimize the processing and communication costs [CHU82]. However, even with a very simplified model of the distributed database and of the cost of processing a query, the costs of computing an optimal solution are large. Experimental work is needed to test the suitability of the model and the real savings before the utility of this research can be evaluated.

Due to the large number of existing DBMS's, a very practical problem is how to map between them or how to provide a single intermediate model which will enable an application to access heterogeneous systems. The research trend seems to be towards higher level non-procedural semantic data models [HAMM81]. There is also interest in integrating database support into the operating system.

3.6 Data Replication And Location Transparency

In order to enhance data availability and reliability, it is desireable to support the replication of critical data at multiple sites. Data replication can be implemented by defining two levels of abstraction [TRAI82a]: database entity --> database objects, and request --> actions. Requests and entities are the elements in the higher level abstraction. Each database entity is physically represented by one or more replicated objects. Requests are implemented by actions on these objects. This separation should be supported by the systems to avoid problems of inconsistency which the user might introduce if updates were improperly done.

The high cost of replication in terms of increased delay and storage requirements makes the fully redundant case impractical in most situations [GARC79b, BARB81]. Many applications would prefer to have those parts of the database which have a high read to update ratio replicated at the sites from which access is frequent. For the majority of data, replication is not expected to be cost effective. Where it is justified, the number of redundant copies will probably be small (two or three copies). A simple partitioned database where the sites and network are reliable should be adequate for many applications.

Knowledge of the physical location of the data should not be required by the end user. This can also be supported by the two level abstraction mechanism [TRAI82a]. However, there is some disagreement over whether the application programmer or the operating system should control the placement of data at a particular site. Also, if the location of data is known, then it should be possible to use this

information to improve performance.

3.7 Nested Transactions

The original transaction concept did not provide the ability to nest subtransactions within a transaction in a hierarchical manner. However, the nested transaction model provides a more flexibile abstraction for many applications [GRAY79]. Since subtransactions may fail independently of the parent, the parent may survive by retrying another subtransaction. Moss [MOSS82], Reed [REED78] and others have proposed ways to extend concurrency control schemes to nested transactions systems, but these have not yet been implemented and evaluated in a general purpose system.

3.8 Distributed Directory/Dictionary Management

General purpose distributed database management systems require a directory/dictionary (catalog) to help manage the database [LIND80a]. The directory/dictionary contains the definitions of the physical and logical structure for the data as well as the rules for mapping between the two. The directory also contains the local names of all resources - relations, files, programs, nodes, etc. - and the addressing rules for locating them. It can be thought of as a specialized distributed database system and as such can be implemented in a wide variety of ways: redundant vs. non redundant copies, centralized vs. partitioned, etc.

The directory/dictionary problem is not unique to DDBMS's. The problems of defining, naming, and locating objects is central to the distributed programming environment [OPPE81] and has been discussed in Section 2.

3.9 Typical Applications

Real-time, interactive transaction processing is the "typical" application for a general purpose DDBMS. But the characteristics and requirements of these systems are vague. How much data is required in the entire database? How much data is required by a single transaction? What percent of the data is located locally vs. remotely? How many sites will a transaction need to access? What percent of transactions are read only vs. update? What is the probability of conflict with other concurrent transactions? The lack of adequate answers to these and many other questions makes it impossible to define a "typical" workload and application.

3.10 Prototypes And Testbeds

The best known DDBMS prototypes are SDD-1 [ROTH80], distributed INGRES [STNE77], and R* [LIND80b]. However, none of these systems meet all of the goals of a DDBMS [TRAI82a] and little has been published regarding the operational performance of these systems.

In order to compare and understand the tradeoffs involved in the design of DDBMS's, a more flexible experimental approach is needed. These systems, called testbeds, emphasize modularity and flexibility so different algorithms and strategies can be easily "plugged in" and

compared. They must also be architecturally similar to real systems in order for the experimental results to be meaningful. The DDTS project at Honeywell [ELMA81] is an example of a planned testbed system for DDBMS experimentation on a wide variety of issues: data models, multi-schema architectures, user interfaces, semantic integrity, data translation, data allocation, transaction optimization, concurrency control, and reliability and recovery. The CARAT project at DEC/UMASS [GARC83] is an operational testbed system where the focus is on the issues of distributed concurrency control, deadlock detection, and crash recovery.

Distributed data management represents an important paradigm for distributed software systems research and development. All the major distributed system research issues, as discussed in the next section, are important to the successful design and implementation of DDBMS's.

4.0 CRITICAL RESEARCH ISSUES

The current state of the art is such that different proposals exist for solving, albeit partially, some of the problems discussed above, but not much is known about their appropriateness, efficiency and applicability in a distributed environment and the tradeoffs that they entail. Hence, there is a need for the following:

1. Theory: A theory of distributed systems needs to be developed in order to deal with issues such as complexity, theoretical limitations, and semantics. Formalisms for distributed computation (both for specifications and for analysis) are required. These

formalisms should be able to handle failure-prone systems also, since error-recovery is one of the crucial aspects of distributed systems.

2. Specification: There is a need to design languages that provide for the specification of a number of features which were, so far, entirely the responsibility of the underlying operating system. These features include data and control distribution, choice of communication primitives, protection requirements and error-recovery schemes.

3. Design, Experimentation and Evaluation: Design methodologies are needed for distributed systems. There is also a need to emphasize the integration of the various solutions to distributed system problems. The experiments should be geared towards building a core of knowledge pertaining to the issues that are relevant to such systems. One of the motivations behind these experiments should be to obtain performance measurements that can be used to evaluate the different proposals. Thus modelling, simulation, and proof techniques should be developed to analyze and evaluate the experimental systems. Obviously, not every solution will be appropriate for all situations. Hence the evaluation of the various proposals should also be directed towards uncovering the assumptions under which a particular scheme performs well. This is especially important in practice, since in the future, distributed systems are bound to be used both for special-purpose as well as general purpose applications.

We now list some of the important open questions that should be addressed in terms of theory, specification, design, experimentation and evaluation.

1. Distribution of Control: Decentralized control algorithms for various functions of operating systems, such as task scheduling and resource allocation, are needed especially those concerned with a high degree of cooperation between decentralized controllers. Investigation of scheduling concepts such as bidding, clustering, co-scheduling, pause time, wave scheduling is required. The use of various mathematical models such as adaptive control, stochastic control, statistical decision theory, and stochastic learning automata for dealing with uncertainty, inaccuracies and delay in distributed systems is also necessary. Scheduling tasks with real time constraints on a loosely coupled distributed system has received little attention to date due to the difficulties involved.

2. Distribution of processes and resources: It is an open question on how to distribute processes that cooperate to execute a given task. This would affect the topology of the resulting network of processes and the manner in which individual nodes are designed. A crucial question is whether movement of processes in execution is worth it and what the best means to implement such movement is. Tradeoffs between static and dynamic allocation of resources should be investigated. Directory assignment, replication and partitioning should also be addressed. For client - server models of distributed file systems where do we divide the responsibility between file servers and clients? When should distributed file systems be

embedded in the operating system and when should a file server model be used? How should the operating system itself be distributed?

3. Protection and security: Specification techniques for the protection requirements within a node and between nodes are needed. Models for protection are required. The problems with revocation of access rights in a distributed environment must be solved. Schemes for implementing protection requirements in a communicating failure-prone environment are needed. Encryption of messages for secure communication and integration of security and protection must be further addressed.

4. Inter process communication: Specification of various aspects of communication paths, such as, directionality, structure of messages, ownership and access rights should be possible. Given the usefulness of different types of message primitives, users should be able to choose and specify the one appropiate for a given situation. Since there are a number of schemes for addressing processes and resources, each requiring differing operating system support and each providing differing levels of transparency and flexibility, the efficacy of the addressing schemes needs further investigation.

5. Error recovery: A more comprehensive theory as well as realistic and practical schemes for error recovery are required. Overheads introduced by fail-safe systems and the trade-offs that such schemes entail must be investigated. How effective is decentralized control as an aid in providing error recovery? Is the nested atomic action the appropriate programming abstraction for the high level

programmer? Can it be efficiently supported? Issues related to network partitioning and slow degradation of the working of nodes in a network is a fairly recent but important research topic.

6. Heterogeneity: Techniques for dealing with information transfer between nodes with varying capabilities, storage schemes, and data models are needed. Efficient network interfaces for heterogenous hosts must be developed. How does heterogeneity affect the distribution of programs and data in a distributed system and vice versa?

7. Synchronization and Concurrency:

In order to implement the atomic action abstraction, some system level mechanism for synchronizing concurrent access requests to shared data must be provided. Concurrency control theory is well developed, and there are many algorithms and approaches for solving the associated distributed deadlock problem assuming locking is used, but there is no sound basis for choosing one approach over another. Experimental studies will be required to compare the performance of alternate approaches for a variety of applications.

References

[ADA80] Reference Manual for the Ada Programming Language", U.S. Department of Defense, July 1980.

[AGRA82] Agrawala, A. K., S. K. Tripathi, and G. Ricart, "Adaptive Routing Using a Virtual Waiting Time Technique," IEEE Transactions on Software Engineering, Vol. SE-8, No. 1, January 1982.

[ANDE71] Anderson, B., et al., "Data Reconfiguration Service," Technical Report, Bolt Beranek and Newman, May 1971.

[ANDR81] Andrews, G.R. "Synchronizing Resources", ACM Transactions on Programming Languages and Systems 3, 4, October 1981, 405-430.

[AOKI78] Aoki, Masanao, "Control of Large-Scale Dynamic Systems by Aggregation," IEEE Transactions on Automatic Control, June 1978.

[APPO81] "Apollo Domain Architecture," Apollo Computer, Inc., February 1981.

[APT80] Apt, K.R., N. Francez, and W. P. De Roever, "A Proof System for Communicating Sequential Processes", ACM Transactions on Programming Languages and Systems 2, 3, July 1980, 359-385.

[BACH79] Bach, Maurice, Nancy Coguen, and Michael Kaplan, "The ADAPT System: A Generalized Approach Towards Data Conversion," Proceedings of the Fifth International Conference on Very Large Data Bases, Rio de Janeiro, Brazil, October 1979.

[BALL76] Ball, J. E., J. Feldman, J. Low, R. Rashid, and P. Rovner, "RIG, Rochester's Intelligent Gateway: System Overview", IEEE Transactions on Software Engineering, Vol SE-2, No. 4, December 1976.

[BALT82] Balter, R., P. Berard, and P. Decitre, "Why Control of Concurrency Level in a Distributed Systems is more fundamental than deadlock Management," ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottowa, Canada, August 1982

[BALZ71] Balzer, R. M., "Ports -- A Method for Dynamic Interprogram Communication and Job Control", Report for an ARPA contract at RAND, August 1971.

[BARB81] Barbara, D. and H. Garcia-Molina, "How Expensive is Data Replication: An Example", Technical Report 286, Dept. of Electrical Engineering and Computer Science, Princeton University, June 1981.

[BASK77] Baskett, F., J. Howard, and J. Montague, "Task Communication in DEMOS", Proceedings of the 6th ACM Symposium on Operating System Principles, November 1977, 23-31.

[BERN79] Bernstein, P.A., D.W. Shipman, and W.S. Wong, "Formal Aspects of Serializability in Database Concurrency Control," IEEE Transactions on Software Engineering, Vol. SE-5, No. 3, May 1979, 203-216.

[BERN80a] Bernstein, P.A., and N. Goodman, "Concurrency Control in Distributed Database Systems," Computing Surveys, Vol. 13, No. 2, June 1981, 185-221.

[BERN80b] Bernstein, P.A., D.W. Shipman, and J.B. Rothnie, Jr., "Concurrency Control in a System for Distributed Databases (SDD-1)," ACM Transactions on Database Systems, Vol. 5, No.

1, March 1980, 18-25.

[BERN81a] Bernstein, P., and N. Goodman, "Concurrency Control in Distributed Database Systems,", ACM Computing Surveys, Vol. 13, No. 2, June 1981.

[BERN81b] Bernstein, P., et. al., "Query Processing in a Distributed Databases (SDD-1)," ACM Tran. on Database Systems, Vol. 6, No. 4, December 1981.

[BERN82a] Bernstein, P., and N. Goodman, "A Sophisticate's Introduction To Distributed Database Concurrency Control," Research Report, TR-19-82, Harvard University, also 8th Intl. Conference on Very Large Data Bases, September 1982.

[BERN82b] Bernstein, P. A. and N. Goodman, "Concurrency Control Algorithms for Multiversion Database Systems," ACM Symposium on Principles of Distributed Computing, Ottowa, Canada, August 1982.

[BERS80] Berstis, V., "Security and Protection of Data in the IBM System/38," Proceedings of the 7th Annual Symposium on Computer Architecture, May 1980.

[BHAR82] Bhargava, B., "Performance Evaluation of the Optimistic Approach to Distributed Database Systems and Its Comparison to Locking," IEEE 3rd Intl. Conf. on Distributed Computed System, October 1982.

[BIRR81] Birrell, Andrew, Roy Levin, Roger Needham and Michael Schroeder, "Grapevine: An Exercise in Distributed Computing," Proceedings of the Eighth Symposium on Operating System Principles, December 1981.

[BOCH78] Bochman, G., "Finite State Description of Communication Protocols", Computer Networks, 2, 1978, 361-372.

[BOKH79] Bokhari, S. H., "Dual Processor Scheduling with Dynamic Reassignment," IEEE Transactions on Software Engineering, Vol. SE-5, No. 4, July 1979.

[BRIN75] Brinch Hansen, P. "The Programming Language Concurrent Pascal", IEEE Transactions on Software Engineering, Vol SE-1 2, June 1975, 199-207.

[BRIN78] Brinch Hansen, P. "Distributed Processes: A Concurrent Programming Concept", Comm. of the ACM, 21, 11, November 1978, 934-941.

[BRIT80] Britton, D. E., and M. E. Stickel, "An Interprocess Communication Facility for Distributed Applications", Proceedings of the 1980 COMPCON Conference on Distributed Computing, February, 1980.

[BRYA81] Bryant, R. M., and R. A. Finkel, "A Stable Distributed

Scheduling Algorithm," Proc of the 2nd International Conference in Distributed Computing Systems, April 1981.

[CASE77] Casey, L. and N. Shelness, "A Domain Structure for Distributed Computer System," Proceedings of the 6th ACM Symposium on Operating Systems Principles, November 1977, 101-108.

[CHAN69] Chandrashekaran and Shen, "Stochastic Automata Games," IEEE Transactions on Systems, Science and Cybernatics, April 1969.

[CHAN79] Chandy, K.M. and J. Misra, "Deadlock Absence Proofs For Networks of Communicating Processes", Information Processing Letters, November 1979, 185-189.

[CHAN81] Chandy, K.M. and J. Misra, "Proofs of Networks of processes", IEEE Transactions on Software Engineering, SE-7, 4, July 1981, 417-426.

[CHAN82] Chandy, K. M. and J. Misra, "A Distributed Algorithm for Detecting Resource Deadlocks in Distributed Systems," ACM Symposium on Principles of Distributed Computing, Ottowa, Canada, August 1982.

[CHNG82] Chang, E. "Echo Algorithms: Depth Parallel Operations on General Graphs", IEEE Transactions on Software Engineering, SE-8, 4, August 1982, 391-400.

[CHER79] Cheriton, D. R., M. A. Malcolm, L. S. Melen, and G. R. Sager, "Thoth, a Portable Real-Time Operating System", Communications of the ACM, Vol. 22, No. 2, February, 1979.

[CHON75] Chong, Chee-Yee, and Michael Athans, "On the Periodic Coordination of Linear Stochastic Systems," Proceedings of the 1975 IFAC, August 1975.

[CHOW79] Chow, Yuan-Chien, and Walter Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," IEEE Transactions on Computers, Vol. C-28, No. 5, May 1979.

[CHOW82] Chow, T. C. K. and J. A. Abraham, "Load Balancing in Distributed Systems," IEEE Transactions in Software Engineering, Vol. SE-8, No. 4, July 1982.

[CHU69] Chu, W. W., "Optimal File Allocation in a Multiple Computing System," IEEE Transactions on Computers, Vol. C-18, October 1969, 885-889.

[CHU80] Chu, W. W., L. J. Holloway, M. Lan, and K. Efe, "Task Allocation in Distributed Pata Processing," Computer, Vol. 13, November 1980, 57-69.

[CHU82] Chu, W.W., and P. Hurley, "Optimal Query Processing for Distributed Database Systems," IEEE Trans. on Computers, Vol. C-31, No. 9, September 1982, 835-850.

[COMP82] Special Issue on Data Flow Systems, IEEE Computer, February 1982.

[COOK79] Cook, R.P., "*Mod--A Language for Distributed Programming", Proceedings of the First International Conference on Distributed Computing Systems, October 1979, 233-241.

[COSS72] Cosserat, D. C., "A Capability Oriented Multi-Processor System for Real-Time Applications," Proceedings of the International Conference on Computer Communications, October, 1972.

[COX81] Cox, G., W. Corwin, K. Lai, and F. Pollack, "A Unified Model and Implementation for Interprocess Communication in a Multiprocessor Environment", Intel Corporation, 1981.

[DAVI81] Davies, D. W., E. Holler, E. D. Jensen, S. R. Kimbleton, B. W. Lampson, G. LeLann, K. J. Thurber and R. W. Watson, Distributed Systems--Architecture and Implementation, Lecture Notes in Computer Science, Vol. 105, Springer-Verlag, NY, 1981.

[DENN66] Dennis, J. and E. Van E., "Programming Semantics for Multiprogrammed Computations", Communications of the ACM, Vol. 9, No. 3, March 1966.

[DENN76] Denning, P.J., "Fault-Tolerant Operating Systems," Computing Surveys, Vol. 8, No. 4, December 1976, 359-390.

[DIO80] Dio, Jeremy, "The Cambridge File Server," ACM Operating System Review, October 1980.

[DOD80] "Reference Manual for the Ada Programming Language", U.S. Department of Defense, July 1980.

[ECKH78] Eckhouse, R.H., Jr., and J.A. Stankovic, "Issues in Distributed Processing - An Overview of Two Workshops," Computer, Vol. 11, No. 1, January 1978, 22-26.

[EFE82] Efe, Kemal, "Heuristic Models of Task Assignment Scheduling in Distributed Systems," IEEE Computer, Vol. 15, No. 6, June 1982.

[ELDE80] El-Dessouki, O. I., and W. H. Huan, "Distributed Enumeration on Network Computers," IEEE Transactions on Computers, Vol. C-29, 818-825.

[ELLI77] Ellis, C., "A Robust Algorithm for Updating Duplicate Databases," Proceedings of the Second Berkeley Workshop on Distributed Management of Data and Computer Networks," Berkeley, California, May 1977, 146-158.

[ELMA81] Elmasri, R., C. Devor, and S. Rahimi, "Notes on DDTS: An Apparatus for Experimental Research in Distributed Database Management Systems," ACM SIGMOD Record, Vol. 11, No. 4, July

1981, 32-49.

[ENSL78] Enslow, P., "What is a Distributed Data Processing System," IEEE Computer, Vol. 11, No. 1, January 1978.

[ENSL80] Enslow, Philip and Timothy Saponas, "Distributed and Decentralized Control in Fully Distributed Processing Systems," Final Technical Report, GIT-ICS-81/82, September 1980.

[ESWA76] Eswaran, K.P., J.N. Gray, R.A. Lorie, and I.L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," Communications of the ACM, Vol. 19, No. 11, November 1976, 624-633.

[FARB73] Farber, D J., et al., "The Distributed Computer System," Proceedings of the 7th Annual IEEE Computer Society International Conference, February 1973.

[FELD79] Feldman, J.A., "High Level Programming for Distributed Computing", Communications of the ACM 22,6, June 1979, 353-368.

[FORS78] Forsdick, H. C., R. E. Schantz,and R. H. Thomas, "Operating Systems for Computer Networks," IEEE Computer, Vol. 11, No. 1, January 1978.

[FRID81] Fridrich, M., and W. Older, "The FELIX File Server," ACM SIGOPS, Proceedings of the Eighth Symposium on Operating System Principles, December 1981, 37-44.

[FU70] Fu, King-Sun, "Learning Control Systems," IEEE Transactions on Automatic Control, April 1970

[GALL77] Gallager, R., "A Minimum Delay Routing Algorithm Using Distributed Computation," IEEE Transactions on Communications, Vol. COM-25, No. 1, January 1977.

[GARC78] Garcia-Molina, H., "Performance Comparison of Two Update Algorithms for Distributed Databases," Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, 1978, 108-119.

[GARC79a] Garcia-Molina, H., "A Concurrency Control Mechanism for Distributed Databases which Uses Centralized Locking Controllers," Proceedings of the 4th. Berkeley Conference on Distributed Data Management and Computer Networks, August 1979, 113-124.

[GARC79b] Garcia-Molina, H., "Performance of Update Algorithms for Replicated Data in a Distributed Database," Ph.D. Dissertation, Stanford University, 1979

[GARC82] Garcia-Molina, H., "Reliability Issues for Fully Replicated Distributed Databases," Computer, Vol. 16, No. 9, September

1982, 34-42.

[GARC83] Garcia-Molina, H., F. Germano, Jr., and W.H. Kohler, "Architectural Overview of a Distributed Software Testbed," Proceedings of the Sixteenth Hawaii Intl. Conf. on System Science, January 1983.

[GELE82] Gelernter, D. and A. J. Bernstein, "Distributed Communication via Global Buffer", Proceedings of the Symposium on Principles of Distributed Computing, August 1982.

[GENT82] Gentleman, W. M., "Message Passing Between Sequential Processes: the Reply Primitive and the Administrator Concept," Software - Practice and Experience, Vol. 11, 1981, 435-466.

[GIFF79a] Gifford, D., "Weighted Voting for Replicated Data," 7th. Symposium on Operating System Principles, December 1979, 150-159.

[GIFF79b] Gifford, D. K., "Violet: An Experimental Decentralized System, Operating Systems Review 13, 5, December 1979.

[GLIG80] Gligor, V.D., and S.H. Shattuck, "On Deadlock Detection in Distributed Systems," IEEE Transactions on Software Engineering, Vol. SE-6, No. 5, September 1980, 435-440.

[GLOR80] Glorioso, Robert M., and Fernando Colon Osorio, Engineering Intelligent Systems, Digital Press, Bedford, MA, 1980.

[GOLD81] Goldberg, Adele et al., Articles on the Smalltalk System, Byte Vol 6, No. 8, August 1981.

[GONZ77] Gonzalez, M. J., "Deterministic Processor Scheduling," ACM Computing Surveys, Vol. 9, No. 3, September 1977, 173-204.

[GOOD79] Good, D.I., R. M. Cohen, and J. Keeton-Williams, "Principles of Proving Concurrent Programs in Gypsy", Proceedings of the Sixth ACM Symposium on Principles of Programmins Languages, January 1979, 42-52.

[GRAH68] Graham, R.M., "Protection in an Information Processing Utility", Communications of the ACM, Vol. 11, No. 5, May 1968, 365-369.

[GRAY75] Gray, J.N., R. A. Lorie, and G. R. Putzolu, "Granularity of Locks in a Shared Database," Proceedings of the Intl. Conference Very Large Database, Sepember 1975, 428-451.

[GRAY76] Gray, J.N., R. A. Lorie, and G. R. Putzolu, ""Granularity of Locks and Degrees of Consistency in a Shared Database," Proceedings of the Modeling in Database Systems, Nijssen, editor, North-Holland, 1976.

[GRAY79] Gray, J.N., "Notes on Data Base Operating Systems," in

Operating Systems: An Advanced Course, R. Bayer, R.M. Graham, and G. Seegmuller, editors, Springer-Verlag, 1979, 393-481.

[GRAY81] Gray J. N., "The Transaction Concept: Virtues and Limitations," VLDB, September 1981, 144-154.

[GUIL82] Guillemont, Marc, "The Chorus Distributed Operating System: Design and Implementation," International Symposium on Local Computer Networks, Florence, Italy, April 1982.

[HALT71] Halter, A. N., and G. W. Dean, Decisions Under Uncertainty, South-Western Pub. Co., Chicago, IL, 1971.

[HAMI78] Hamilton, J., "Functional Specification for the WEB Kernel," Digital Equipment Corporation, R & D Group, Maynard, MA, November 1978.

[HAMM78] Hammer, M., and D. Shipman, "An Overview of Reliability Mechanisms for a Distributed Data Base System," Proceedings of COMPCON Spring '78, February 1978, 63-65.

[HAMM80] Hammer M. and D. Shipman, "Reliability Mechanisms for SDD-1 : A system for Distributed Databases." ACM Tran. on Database Systems, December 1980.

[HAMM81] Hammer, M., and D. McLeod, "Database Description with SDM: A Semantic Database Model," ACM Trans. on Database Systems, Vol. 6, No. 3, September 1981, 351-386.

[HERL82] Herlihy, M. and B. Liskov, "A Value Transmission Method for Abstract Data Types", ACM Transactions on Programming Languages and Systems, Vol. 4, 4, October 1982, 527-551.

[HO80] Ho, Y., "Team Decision Theory and Information Structures," Proceedings of the IEEE, Vol. 68, No. 6, June 1980.

[HOAR74] Hoare, C.A.R., "Monitors: an Operating System Structuring Concept", Communications of the ACM, 17, 10, October 1974, 549-557.

[HOAR78] Hoare, C.A.R., "CSP: Communicating Sequential Processes", Communications of the ACM, Vol. 21, No. 8, August 1978.

[IRAN79] Irani, K. B. and H. Lin, "Queueing Network Models for Concurrent Transaction Processing in a Database System," SIGMOD 1979.

[ISLO80] Isloor, Sreekaanth, and T. Anthony Marsland, "The Deadlock Problem: An Overview," IEEE Computer, Vol. 13, No. 9, 1980.

[JARV75] Jarvis, R. A., "Optimization Strategies in Adaptive Control: A Selective Survey," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SME-5, No. 1, January 1975.

[JENS78] Jensen, E. D., "The Honeywell Experimental Distributed Processor-An Overview of its Objective, Philosophy and Architectural Facilities," IEEE Computer, Vol. 11, No. 1, January 1978.

[JONE78a] Jones, A.K., "The object model: a conceptual tool for structuring software" Lecture Notes in Computer Science 60, Springer-Verlag, 1978, 3-19.

[JONE78b] Jones, A.K. and B. Liskov, "A Language Extension for Expressing Constraints on Data Access", Communications of the ACM, Vol. 21, 5, May 1978, 358-367.

[JONE79] Jones, A. K., R. J. Chansler, I. Durham, K. Schwans, and S. R. Vegdahl, "StarOS, a Multiprocessor Operating System for the Support of Task Forces", Proceedings of the 7th Symposium on Operating System Principles, December 1979.

[KAHN81] KAHN, K. C., et al., "iMax: A Multiprocessor Operating System for an Object-Based Computer," Proceedings of the Eighth Symposium on Operating System Principles, December 14-16, 1981.

[KAMO82] Kamoun, F., M. B. Djerad, and G. Le Lann, "Queueing Analysis of the Ordering Issue in a Distributed Database Concurrency Control Mechanism: a General Case," IEEE 3rd Intl. Conf. on Distributed Computed System, October 1982.

[KIMB78] Kimbleton, S. R., H. M. Wood, and M. L. Fitzgerald, "Network Operating Systems--An Implementation Approach," Proceedings of the AFIPS Conference, 47, 1978.

[KLEI81] Kleinrock, L., and A. Nilsson, "On Optimal Scheduling Algorithms for Time-Shared Systems," JACM, Vol. 28, No. 3, July 1981, 477-486.

[KNOT75] Knott, "A Proposal for Certain Process Management and Intercommunication Primitives", Operating Systems Review, October and January, 1974-75.

[KOHL81] Kohler, W. H., "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," ACM Computing Surveys, Vol. 13, No. 2, June 1981, 149-183.

[KORT82] Korth, H. T., "Edge Locks and Deadlock Avoidance in Distributed Systems," ACM SIGACT- SIGOPS Aymp. on Principle of Distributed Computing, Ottowa, Canada, August, 1982, 173-182.

[KUNG81] Kung, H. T. and J. T. Robinson, "On Optimistic Methods for Concurrency Control," ACM Tran. on Database Systems, Vol. 6, No. 2, June 1981.

[LAMP76] Lampson, B. W., and H. E. Sturgis, "Reflections on an Operating System Design", Communications of the ACM, Vol. 19,

No. 5, May 1976.

[LAMP79] Lampson, B.W., and H.E. Sturgis, "Crash Recovery in a Distributed Storage System," revised and expanded unpublished paper, Computer Science Laboratory, Xerox Palo Alto Research Center, Palo Alto, California 94304, 1979.

[LANT80] Lantz, K. A., "RIG, An Architecture for Distributed Systems" Proceedings of the ACM Pacific 80, November 1980.

[LARS79] Larsen, R. E., "Tutorial: Distributed Control," IEEE Catalog No. EHO 153-7, IEEE Press, New York, 1979.

[LAZO81] Lazowska, E., H. Levy, G. Almes, M. Fischer, R. Fowler, and S. Vestal, "The Architecture of the Eden System", 8th Annual Symposium on OS Principles, December 1981.

[LELA78] Le Lann, G., "Algorithms for Distributed Data-Sharing Systems Which Use Tickets," Proceedings of the 3rd Berkeley Workshop on Dist. Databases and Comp. Network, 1978.

[LELA80] LeLann, G., "Distributed Systems-Towards a Formal Approach," Proceedings of IFIP Congress, Toronto, North Holland Pub., August 1980, 155-160.

[LELA81] LeLann, G., "A Distributed System for Real-Time Transaction Processing," IEEE Computer, Vol. 14, No. 2, February 1981.

[LESS80] Lesser, Victor, Daniel Serrain, and Jeff Bonar, "PCL: A Process-Oriented Job Control Language," IEEE Transactions on Computers, Vol. C-29, No. 12, December 1980.

[LEVI77] Levine, Paul H., "Facilitating Interprocess Communication in a Heterogeneous Network Environment," Masters Thesis, MIT, June 1977.

[LEVY81] Levy, H., "A Comparative Study of Capability-Based Computer Architectures", University of Washington Master's Thesis, October 1981.

[LIN79] Lin, W. K., "Concurrency Control in a Multiple-Copy Distributed Database System," 4th Berkeley Conference on Distributed Data management and Computer Network, August 1979.

[LIN81] Lin, W. K., "Performance Evaluation of Two Concurrency Mechnisms in a Distributed Database System," SIGMOD 1981.

[LIND79] Lindsay B., et. al., "Notes on Distributed Databases," IBM Research Report, Report RJ2571, 1979.

[LIND80a] Lindsay, B., "Object Naming and Catalog Management for a Distributed Database Manager," IBM Research Report, RJ2914 (36689), August 29, 1980.

[LIND80b] Lindsay, B., and P.G. Selinger, "Site Autonomy Issues in R*:

A Distributed Database Management System," IBM Research Report, RJ2927 (36822), September 15, 1980.

[LIND81] Lindgren, B. W., Elements of Decision Theory, The MacMillan Co., NY, 1981.

[LISK75] Liskov, B. and S. Zilles, "Specification Techniques for Data Abstractions", IEEE Trans. on Software Engineering, Vol. 1, No. 1, March 1975, 7-18.

[LISK79] Liskov, Barbara, "Primitives for Distributed Computing", Proceedings of the 7th Symposium of Operating System Principles, December 1979.

[LISK82] Liskov, B. and R. Scheifler, "Guardians and Actions: Linguistic Support for Robust, Distributed Programs", Proceedings of the Ninth Symposium on Principles of Programming Languages, January 1982, 7-19.

[LOME77] Lomet, D.B., "Process Structuring, Synchronization, and Recovery Using Atomic Actions," Proceedings of the ACM Conference on Language Design for Reliable Software, SIGPLAN Notices, Vol. 12, No. 3, March 1977, 128-137.

[LUDE81] Luderer, G. W. R., et al., "A Distributed UNIX System Based on a Virtual Circuit Switch," Proceedings of the Eighth Symposium on Operating System Principles, December 14-16, 1981.

[MA82] Ma, P. Y. R., E. Y. S. Lee, and J. Tsuchiya, "A Task Allocation Model for Distributed Computing Systems,"IEEE Transactions on Computers, Vol. C-31, No. 1, January 1982, 41-47.

[MCQU74] McQuillan, J. M., "Adaptive Routing Algorithms for Distributed Computer Networks," BBN Report 2831, May 1974.

[MELL81] Melliar-Smith, P.M. and R. L. Schwartz, "Temporal Logic Specifications of Distributed Systems", Proceedings of the of the Second International Conference on Distributed Systems, April 1981.

[MENA79] Menasce, D. A. and R. R. Muntz, R., "Locking and Deadlock Detection in Distributed Data Bases," IEEE Tran. on Software Engineering, Vol. SE-5, No. 3, May 1979

[MENA80] Menasce, D. A., G. J. Popek, and R. R. Muntz, "A Locking Protocol for Resource Coordination in Distributed Database," ACM Tran. on Database Systems, Vol. 5, No. 2, June 1980.

[MINO82] Minoura, T. and G. Wiederhold, "Resilient Extended True-Copy Token Scheme for a Distributed Database System," IEEE Tran. on Software Engineering, Vol. SE-8, No. 3, May 1982.

[MITC79] Mitchel, J.G., W. Maybury, and R. Sweet, "Mesa Language Manual", Xerox PARC Report CSL-79-3, April 1979.

[MOSS82] Moss, J. E. B., "Nested Transactions and Reliable Distributed Computing," Second Symposium on Reliability in Distributed Software and Database Systems, July, 1982.

[NARE74] Narendra, K., "Learning Automata - A Survey," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-4, No. 4, July 1974.

[NEED77] Needham, R. M., and R. D. Walker, "The Cambridge CAP Computer and its Protection System", Proceedings of the 6th ACM Symposium on Operating System Principles, November 1977.

[NELS81] Nelson, B. J., "Remote Procedure Call", Xerox Corperation Technical Report CSL-81-9, May, 1981.

[OBER82] Obermarck, R., "Distributed Deadlock Detection Algorithm," ACM Tran. on Database Systems, Vol. 7, No. 2, June 1982.

[OPPE81] Oppen, D.C., and Y.K. Dalal, "The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment," Xerox Corporation, Office Products Division Report OPD-T8103, October 1981.

[OUST80] Ousterhout, J., D. Scelza, and P. Sindhu, "Medusa: An Experiment in Distributed Operating System Structure", Communications of the ACM, Vol. 23, No. 2, February 1980.

[PASH82] Pashtan, A., "Object oreinted operating systems: An emerging design methodology", Proceedings of the of the ACM National Conference, October 1982, 126-131.

[POPE81] Popek, G., et al., "LOCUS, A Network Trans. Parent, High Reliability Distributed System" Proceedings of the Eighth Symposium on Operating System Principles, December 14-16, 1981.

[POWE77] Powell, M., "The DEMOS File System", Proceedings of the 6th ACM Symposium on Operating System Principles, November 1977, 33-42.

[RAIF61] Raiffa, H., and R. Schlaifer, Applied Statistical Decision Theory, Division of Research, Graduate School of Business Adm., Harvard University, Cambridge, MA, 1961.

[RAMA82] Ramamritham, K., "Proof Techniques for Resource Controllers", COINS Technical Report 82-18, University of Massachusetts, January 1982.

[RASH80] Rashid, R., "An Inter-Process Communication Facility for UNIX", Carnegie-Mellon University Technical Report, June 1980.

[RASH81] Rashid, R. F., and G. G. Robertson, "Accent: A Communication Oriented Network Operating System Kernel," Proceedings of the Eighth Symposium on Operating System Principles, December 14-16, 1981.

[RAO80] Rao, Ram, "Design and Evaluation of Distributed Communication Primitives", ACM Pacific 1980, November, 1980.

[REED78] Reed, D. P., "Naming and Synchronization in a Decentralized Computer System," Ph.D. Dissertation, MIT, 1978.

[REED79] Reed, D.P., "Implementing Atomic Actions on Decentralized Data," Proceedings of the 7th. ACM Symposium on Operating System Principles, December 1979.

[RIES79a] Ries D., "The Effects of Concurrency Control on the Performance of a Distributed Data Management System," Proceedings of the 4th Berkeley Workshop on Distributed Data Management and Comp. Networks, 1979.

[RIES79b] Ries, D. R., and M. R. Stonebraker, "Locking Granularity Revisited," ACM Tran. on Database System, June 1979, 210-227.

[RITC74] Ritchie, D. and K. Thompson, "The UNIX Time-Sharing System", Communications of the ACM, Vol. 17, No. 7, July 1974.

[ROSE78] Rosenkrantz, D.J., R.E. Stearns, and P.M. Lewis, "System Level Concurrency Control for Distributed Database Systems," ACM Transactions on Database Systems, Vol. 3, No. 2, June 1978, 178-198.

[ROTH77] Rothnie, J.B,, and N. Goodman, "A Survey of Research and Development in Distributed Database Management," Proceedings of the Third Intl. Conference on Very Large Data Bases, 1977, 48-62.

[ROTH80] Rothnie, J.B., Jr., P.A. Bernstein, S. Fox, N. Goodman, M. Hammer, T.A. Landers, C. Reeve, D.W. Shipman, and E. Wong, "Introduction to a System for Distributed Databases (SDD-1)," ACM Transactions on Database Systems, Vol. 5, No. 1, March 1980, 1-17.

[ROWE79] Rowe, L. A. and K. P. Birman, "Network Support for a Distributed Database System," Proceedings of the 4th Berkeley Conference on Distributed Data Management and Computer Networks, 1979.

[ROWE82] Rowe, L. A., and K. P. Birman, "A Local Network Based on the UNIX Operating System," IEEE Transactions on Software Engineering, Vol. SE-8, No. 2, March 1982.

[RUSS80] Russ[illegible] D.L. "State Restoration in Systems of Communicating Processes", IEEE Transactions on Software Engineering, March 1980, 183-194.

[RYPK79] Rypka, D.J., and A.P. Lucido, "Deadock Detection and Avoidance for Shared Logical Resources," IEEE Transactions on Software Engineering, Vol. SE-5, No. 5, September 1979, 465-471.

[SALT75] Saltzer, J.H. and M. D. Schroeder, "The Protection of Information in Computer Systems", Proceedings of the IEEE, Vol. 63, No. 9, September 1975, 1278-1308.

[SALT78] Saltzer, J. H., "Naming and Binding of Objects," Operating Systems: An Advanced Course, Springer-Verlag, 1978.

[SAND78] Sandell, N., R. Varaiya, M. Athans, and M. Safonov, "Survey of Decentralized Control Methods for Large Scale Systems, IEEE Transactions on Automatic Control, Vol. AC-23, No. 2, April 1978.

[SCHO78] Schoch, J. F., "Inter-Network Naming, Addressing and Routing," Compcon 78, Spring 1978.

[SCHL81] Schlageter, G., "Optimistic methods For Concurrency Control in Distributed Databse System," VLDB 1981.

[SEGA77] Segall, A., "The Modelling of Adaptive Routing in Data-Communication Networks," IEEE Trans. on Communications, Vol. COM-25, No. 1, January 1977, 85-95.

[SKEE81] Skeen, D. and M. Stonebraker, "A Formal Model of Crash Recovery in a Distributed System," Proceedings of the Fifth Berkeley Workshop on Distributed Data Management and Computer Networks, February 1981, 129-142.

[SMIT80] Smith, G. R., "The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver," IEEE Transactions on Computers, Vol. C-29, No. 12, December 1980.

[SOLO79] Solomon, M. H., and R. A. Finkel, "The Roscoe Distributed Operating System", Proceedings of the 7th Symposium on Operating System Principles, March 1979.

[STAN79] Stankovic, J. A., and A. van Dam, "Research Directions in (Cooperative) Distributed Processing," Research Directions in Software Technology, MIT Press, Cambridge, MA, 1979.

[STAN81] Stankovic, John A., "ADCOS - An Adaptive, System Wide, Decentralized Controlled Operating System," Univ. of Massachusetts, Technical Report, November 1981.

[STAN82a] Stankovic, John A., "Software Communication Mechanisms: Procedure Calls Versus Messages," IEEE Computer, Vol. 15, No. 4, April 1982.

[STAN82b] Stankovic, J. A., N. Chowdhury, R. Mirchandaney, I. Sidhu, "An Evaluation of the Applicability of Different Mathematical Approaches to the Analysis of Decentralized Control Algorithms, Proceedings of COMPSAC 82, November 1982.

[STAN82c] Stankovic, John A., "A Stochastic Model For Replicated Decentralized Control," submitted to IEEE Transactions on

Computers, December 1982.

[STAN83a] Stankovic, J. A., "A Heuristic for Cooperation Among Decentralized Controllers," Proceedings of INFOCOM 83, April 1983.

[STAN83b] Stankovic, J. A., "Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms," submitted to Computer Networks, January 1983.

[STEM82] Stemple, D., K. Ramamritham, and S. Vinter, "Preliminary Design of a Port-based Operating System", COINS Technical Report 82-24, Dept. of Computer and Information Sciences, University of Mass., October 1982.

[STON77] Stone, H. S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," IEEE Trans. on Software Engineering, Vol. SE-3, January 1977.

[STON78a] Stone, H. S., "Critical Load Factors in Distributed Computer Systems," IEEE Trans. on Software Engineering, Vol. SE-4, May 1978.

[STON78b] Stone, H. S. and S. H. Bokhari, "Control of Distributed Processes," IEEE Computer, Vol. 11, No. 7, July 1978, 97-106.

[STNE77] Stonebraker, M., and E. Neuhold, "A Distributed Database Version of INGRES," Proceedings of the 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, 19-36.

[STNE79] Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," IEEE Transactions on Software Engineering, Vol. SE-5, No. 3, May 1979, 180-194.

[STNE81] Stonebraker, M., "Operating System Support for Database Management," Communications of the ACM, Vol. 24, No. 7, July 1981, 412-418.

[STUR80] Sturgles, H., J. Mitchell, and J. Israel, "Issues in the Design and Use of a Distributed File System," ACM Operating System Review, July 1980, 55-69.

[SUNS77] Sunshine, C., "Interprocess Communication Extensions for the UNIX Operating System: Design Considerations", Rand Corperation Publication R-2064/1-AF, June 1977.

[SWAN77] Swan, R. J., S. H. Fuller, and D. P. Siewiorek, "Cm: A Modular, Multi-Microprocessor," AFIPS Conference, Vol. 46, NCC, 1977.

[TEN81a] Tenney, R. R., and N. R. Sandell, Jr., "Structures for Distributed Decision-making," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, No. 8, August 1981,

517-527.

[TEN81b] Tenney, R. R., and N. R. Sandell, Jr., "Strategies for Distributed Decisionmaking," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, No. 8, August 1981, 527-538.

[THOM79] Thomas, R.H., "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases," ACM Transactions on Database Systems, Vol. 4, No. 2, June 1979, 180-209.

[TRAI82a] Traiger, I., et. al., "Transactions and Consistency in Distributed Database System," ACM Tran. on Database System, Vol 7, No. 3, September 1982.

[TRAI82b] Traiger, I., "Virtual Memory Management for Database Systems," ACM Operating System Review, Vol. 16, No. 4, October 1982, 26-48.

[VERH78] Verhofstad, J.S.M., "Recovery Techniques for Database Systems," Computing Surveys, Vol. 10, No. 2, June 1978, 167-195.

[VINT83] Vinter, S., K. Ramamritham, and D. Stemple, "Protecting Objects through the use of Ports", Proceedings of the Phoenix Conference on Computers and Communication, March 1983.

[WALD72] Walden, David C., "A System for Interprocess Communication in a Resource Sharing Computer Network", Communications of the ACM, Vol. 15, No. 4, April 1972.

[WARD80] Ward, S., "TRIX; A Network Oriented Operating System," Proceedings COMPCON, 1980.

[WILK79] Wilkes, M. V., and R. M. Needham, "The Cambridge CAP Computer and its Operating System," Elsevier North Holland, 1979.

[WINK72] Winkler, R. L., Introduction to Bayesian Inference and Decision, Holt, Rindhard & Winston, Inc., NY, 1972.

[WIRT77] Wirth, N., "Modula: a Language for Modular Multiprogramming", Software - Practice and Experience, 7,1, January 1977, 3-35.

[WITT80] Wittie, L., and Andre M. Van Tilborg, "MICROS, A Distributed Operating System for Micronet, A Reconfigurable Network Computer, IEEE Transactions on Computers, Vol. C-29, No. 12, December 1980.

[WULF74] Wulf, W., E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, "HYDRA: The Kernel of a Multiprocessor Operating System", Communications of the ACM, Vol. 17, No. 6, June 1974.

Simulations of Three Adaptive, Decentralized Controlled, Job Scheduling Algorithms

by

John A. Stankovic
413-545-0720

January, 1983

Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, Mass. 01003

This work was supported, in part, by the National Science Foundation under grant MCS-8104203, and by the US Army CECOM, CENCOMS under grant number DAAB07-82-K-J015.

Abstract

Simulation results of three adaptive, decentralized controlled job scheduling algorithms are presented. The results provide insight into the workings and relative effectiveness of the three algorithms, as well as insight into the performance of a special type of decentralized control. The simulation approach includes tuning the parameters of each algorithm, and then comparing the three algorithms based on response time, load balancing and the percentage of job movement. Each of the algorithms is compared under light, moderate, and heavy loads in the system, as well as a function of the traffic in the communication subnet and the scheduling interval. Three simple analytical models are also presented and compared to the simulation results. A general observation is that, if tuned correctly, the decentralized algorithms exhibit stable behavior and considerably improve performance (response time and load balancing) at modest cost (percentage of job movement). Overall, the results contribute to the understanding of a special type of decentralized control algorithm that has not been extensively studied but is becoming more and more important.

KEYWORDS: Adaptive algorithms, decentralized control algorithms, decentralized control, distributed processing, job scheduling, networking and simulation.

1.0 INTRODUCTION

A distributed processing system is defined as a collection of processor-memory pairs (hosts) that are physically and logically interconnected, with decentralized system-wide control of all resources, for the cooperative execution of application programs [10], [14]. Such systems may be dedicated to a single application or may implement a general purpose computing facility. By decentralized system-wide control is meant that there exists distributed resources in the system, that there is decentralized control of these resources (i.e., there is no single, central host "in charge," nor is there a central state table), and that there is system-wide cooperation between independent hosts which results in a single unified system. By system-wide cooperation is meant that the algorithms of the system operate for the "good of the whole" and not for a particular host. For systems meeting this restrictive definition of distributed processing, it is hypothesized that their reliability, extensibility, and performance will be better than what is generally available today. In this paper the term distributed processing refers to this very specific type of highly integrated distributed system.

For distributed processing systems to achieve their potential, questions relating to the effectiveness and stability of decentralized control algorithms must be answered. Solutions to the decentralized control problem in distributed databases are known. In this type of decentralized control problem, decentralized controllers must cooperate to achieve a system-wide objective of good performance subject to the data integrity constraint. The cooperation is achieved in various ways, e.g., either by the combined principles of atomic actions and unique timestamps or by the combined priciples of atomic actions and two-phase locking. It can then be proven that multiple decentralized controllers can operate concurrently and still meet the data integrity constraint.

Most other research on decentralized control is better described as research on decomposition techniques rather than decentralized control. In such work, large scale problems are partitioned into smaller problems, each smaller problem being solved, for example, by mathematical programming techniques, and the separate solutions being combined via a few interaction variables. See [16] for an excellent summary of these types of decentralized control (decomposition).

Our interests lie in a type of decentralized control we refer to as stochastic replicated decentralized control [25]. By replicated we mean that the decentralized entities (controllers) implementing the function are involved in the entire problem, not just a subset of it. This type of decentralized control is appropriate for certain functions like job scheduling where data integrity is not crucial.

Mathematical treatment [2], [13], [20], [24], [30], [31] of decentralized control has not provided answers for stochastic replicated functions of distributed processing systems. This is due to the mathematically intractable nature of the problem when all simultaneously complicating factors are taken into account [22], [24], [25]. These factors include multiple, concurrent, decentralized controllers, each with equal authority, each working with noisy and out of date information; further, there are unpredictable future events that can occur including significant delays, and decisions must be made quickly. Lacking analytical results, simulation can provide insight into the operation and effectiveness of this special type of decentralized control algorithm (a stochastic replicated function) as part of distributed processing systems.

In this paper simulation results of three adaptive, decentralized controlled job scheduling algorithms are presented. We try to answer the questions of whether multiple, decentralized, replicated, concurrently executing controllers can act together in such a manner so as to produce greater benefits than costs, and whether concurrent actions taken by the multiple controllers produce stable behavior? The simulation approach includes tuning each algorithm, and then comparing the three based on response time, load balancing, and the percentage of job movement. By tuning is meant that values for certain parameters of the algorithm are chosen to improve the performance of the algorithm. Each of the algorithms is compared under light, moderate, and heavy loads in the system, as well as a function of the traffic in the commmunication subnet and the scheduling interval. Three simple analytical models are also presented and compared to the simulation results. A general observation is that, if tuned correctly, the algorithms exhibit stable behavior and considerably improve performance (response time and load balancing) at modest cost (percentage of job movement). Equally important is that the algorithms are very simple and incur negligible run time costs. Conversely, if the algorithms are not tuned correctly they do not exhibit stable behavior and do incur high cost.

Section 2 describes exactly what is meant by a stochastic, replicated, decentralized controlled job scheduling algorithm, itemizes some special concerns of a practical algorithm for a distributed processing system, and relates this scheduling research to other current work. In the remainder of this paper we drop the terms stochastic and replicated for convenience but these characteristics are always implied. Section 3 presents the basic simulation model used in this study. Section 4 describes the three algorithms to be compared as well as the motivations behind the choice of these algorithms. Section 5 presents and discusses the results of the simulations, including comparisons to analytical results. Section 6 contains a summary of the results of the paper.

2.0 DECENTRALIZED CONTROLLED JOB SCHEDULING

A decentralized controlled job scheduling algorithm is not the typical scheduling algorithm one finds in the literature. Its operation and environment are somewhat unique. This section describes those aspects of a decentralized controlled job scheduling algorithm and its environment that make it unique, along with its relationship to other scheduling research.

A decentralized controlled job scheduling algorithm is one algorithm composed of "n" physically distributed entities, $\{e_1, e_2, \ldots, e_n\}$. Each of the entities is considered a local controller. Each of these local controllers runs asynchronously and concurrently with the others, continually making decisions over time. Each entity e_i makes decisions based on a system-wide objective function, rather than on a local one. Each e_i makes decisions on an equal basis with the other entities (there is no master entity, even for a short period of time!). It is intended that the job scheduling algorithm adapts to the changing busyness of the various hosts in the system.

The environment in which the job scheduling entities are running is stochastic in two ways. First, the observations entities make about the state of the system are uncertain (they are estimates). Second, once a decision is made (e.g., move a job to host i), future random forces (e.g., a burst of jobs arrive at host i) that are independent of the control decision can occur.

In general, the observations themselves can be made in any number of ways. In this paper it is assumed that each entity periodically transmits its view of the busyness of the system to its neighbors, and upon receiving updates from its neighbors combines those updates in some manner so as to obtain its new view of the system. The update scheme used in our simulation is described in the next section.

Since job scheduling is an operating system function, any algorithm implementing job scheduling must run quickly. This is an extremely important aspect of the algorithms and makes many potential solutions unsuitable. For example, modeling the system by a mathematical program and solving it on-line is out of the question.

The execution costs involved in running a decentralized controlled job scheduling algorithm include the cost of running the algorithm itself, the cost of transmitting update information, and the cost of moving jobs. The primary goal of the algorithm is to minimize response time with minimum job movement. The secondary goal is to balance the load. All of these costs and goals are addressed with our simulations.

It is important to note the differences between job scheduling as studied in this paper and task allocation research. The task allocation problem for distributed systems normally assumes that all the tasks to be allocated are known beforehand. The cost of running each task on each processor is also assumed known. Further, all intertask communication patterns, as well as the cost of communicating over the network are assumed known. Given these assumptions, there are graph-theoretic [3], [28], [29], integer programming [6], [7], [9] queueing theoretic [1], [5], [15] and heuristic methods [4], [8], [18] for dealing with the task allocation problem. For a more detailed survey of some of these techniques see [12].

In other scheduling research based on the bidding scheme [11], [21] specific tasks are matched to processors based on the current ability of the processors to perform this work. These schemes are suboptimal, but are more extensible and adaptable than many of the other approaches. However, the cost of making and acquiring bids may become excessive, and the factors used in making the bids have not been extensively studied.

Wave scheduling [32] and co-scheduling [19] are concepts that apply to clustering related jobs onto the same host and is not considered in our research because it deals with process scheduling and assumes prior knowledge.

The job scheduling problem for a general purpose distributed processing system is different from the above referenced works. Jobs arrive at each geographically distributed host in some unpredictable manner. There is no central queue. At no point in time does the decentralized job scheduler know all the jobs to be assigned. Characteristics of the incoming jobs are also unknown. That is, the processing costs of the jobs, whether the job is actually multiple tasks that communicate with each other, etc., are all unknown. The job scheduler's task is to provide a gross level of load balancing, hoping to improve response time at low cost. In other words, a practical job scheduling algorithm must be simple, run fast, minimize job movement and improve system-wide response time and load balancing. Given that this high level load balancing of jobs is performed, a decentralized process scheduler can also be implemented that deals with processes (jobs in execution), interprocess communication, clustering and co-scheduling of processes. Process scheduling is not treated in this paper.

Work similar to the work described here, except that statistical decision theory is used as an integral part of the algorithms, can be found in [23], [26], and [27]. In such work more sophisticated scheduling algorithms are employed but the additional benefit of that sophistication is yet to be proven conclusively.

The motivations for splitting the scheduling function into two parts (job and process scheduling) are:

1. the job scheduler can be simple and therefore it is possible to study the use of decentralized control algorithms in a simpler situation (in this work we are attempting to determine how multiple, interacting, decentralized components of a single function interact), and,

2. to eventually study how two (or more) decentralized control algorithms interact with each other, e.g., interaction between the job scheduler and the process scheduler may provide insight into how an entire system composed of multiple decentralized control algorithms of various types might function.

3.0 THE SIMULATION MODEL

The simulation model, programmed in GPSS, consists of a network of five hosts connected as shown in Figure 1. The unit of time in the simulation is milliseconds. Each host is considered identical except for processor speed. The service time of a job scheduled for execution is chosen from an exponential distribution with different averages for each host. The averages are 5000, 7000, 6000, 5000, and 7000 milliseconds for hosts 1-5 respectively. There are five independent sources for arrivals of jobs. Each source is modeled by a Poisson distribution with averages $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$. The λ's vary depending on the particular loads (light, moderate, heavy) being modeled. Specific values of the λ's are presented with the simulation results in Section 5. When a job arrives at a host from the external world, it is assigned a size based on the distribution shown in Figure 2. Delay in the communications subnet is modeled as a simple function, i.e., the size of the information to be sent divided by the packet size (1K bits) times the average delay per packet. Hence, the delay in the subnet is independent of the topology in our simulations. Both jobs and state information are passed into the subnet, thereby modeling two of the major costs involved. The third major cost, the cost of running the algorithm is modeled as a fixed cost of 50 milliseconds each time it runs on each host.

In the simulation, each host periodically calculates an estimate of the number of jobs at each host in the network, and sends this information to its nearest neighbors. This state information is updated at each host in the following way. Consider three classes of hosts, myself, my neighbors, and all others. In general, host i can determine precisely the number of jobs in its own queue (accurate local data), and therefore, will believe its own value rather than his

neighbors perception of his workload. Since the nearest neighbors are only one hop away, their estimates of their workload, as passed to host i, will be only slightly out of date and, in general, will be a better estimate than estimates other nodes have of them. Therefore, host i uses the nearest neighbors estimate of themselves. Finally, all other views of any remaining hosts in the network are determined by taking an average of the estimates from all the incoming update messages. Between updates no attempt is made at estimating either the current state since the last update nor any future state. Old information is simply used. We believe that the additional cost of such estimates is prohibitive in comparison to the potential benefits when the update interval is frequent (as in our case). Estimates between updates would be beneficial if (a) a reasonable estimation rule were known, and (b) the state information update interval were relatively infrequent.

The model also includes a simple FCFS process scheduler that does not allow multiprogramming. A statistics gathering capability sufficient to determine the average response time per job, average queue length per host, and the number of jobs moved per host is included in the model. The model was designed so that it is also easy to plug-in different job scheduling algorithms. Three such algorithms, and the aspects of the simulation model relating to these algorithms, are described in the next section. All simulations are run for 10 minutes of simulated time, statistics are cleared, and then run for 30 more minutes. This is done to minimize the transient start up effects of an empty system. Longer runs did not substantially alter the results so to save computer time a total of 40 minutes per run was chosen as the test standard.

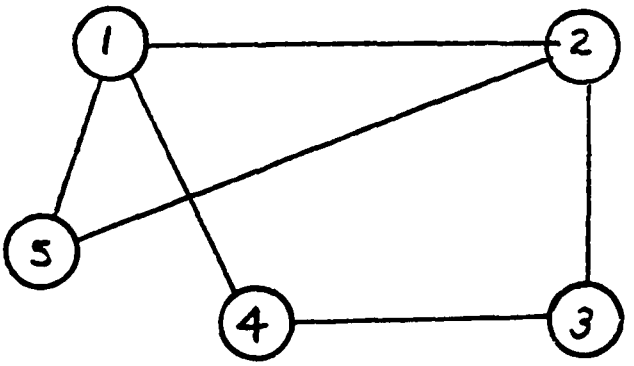


Figure 1: Network Topology

.1	1000	.85	22000
.2	12000	.90	30000
.4	14000	.95	34000
.6	16000	.98	38000
.7	18000	.99	44000
.8	20000	.995	50000

Figure 2: Job Size Distribution

4.0 THE JOB SCHEDULING ALGORITHMS

This section describes the three job scheduling algorithms that are implemented and compared via simulation. Before the algorithms themselves are described, those aspects of the simulation that are common to the three algorithms are described.

In each of the algorithms, the job scheduling entities are activated periodically (initially every 2 seconds and then varied for several tests) and also whenever the process scheduler of a host needs to activate a new job for execution. Furthermore, state information (in this case, the number of jobs) about the busyness of the network is passed between neighbors on a periodic basis (initially every 2 seconds and then varies with the scheduling interval in later tests). Each scheduling entity then has an out of date observation of how many jobs exist at each site in the network. In future simulations we intend to study the effect of using more sophisticated state information (not just the number of jobs) to estimate the busyness of a host, and to use asynchronous updates of state information rather than periodic.

A practical consideration for job scheduling algorithms comes into play for very lightly loaded and very heavily loaded systems. In both instances it is not beneficial to move jobs; therefore, in algorithms 1 and 3, described below, jobs are not moved by a host if it observes a very lightly or very heavily loaded system. Very lightly loaded is defined as "each host has less than 4 jobs." A very heavily loaded system is defined as "each host has more than 20 jobs." All other situations are considered moderate loads. We did not include these cutoffs in algorithm 2 in order to test the effect of omitting such cutoffs.

Another situation that is common to these algorithms is related to the delay in the communication subnet. If the delay in moving jobs is large compared to the periodic update rate of scheduling entities, then the scheduling entities may continue to send jobs to a host not realizing there are many jobs "on the way" (in the subnet) to that host. There are a

number of ways to deal with this problem. Our approach in algorithms 1 and 2 is to choose a fairly slow periodic update rate (acceptable for job scheduling) and move at most 2 jobs to a host at one time, so that even if there is a substantial delay in the subnet it is not expected that many jobs will be "on the way." However, in algorithm 3 we keep track of which host has been sent jobs recently and use this information in an effort to mitigate the problem and to minimize job movement.

Algorithm 1 For moderate loads in the system, each entity compares its own busyness to its observation (estimate) of the busyness of the least busy host. Note that the host thought to be least busy is itself an estimate. The difference between the busyness of these two hosts is then compared to a bias. If the difference is less than the bias, then no job is moved, else, one job is moved to the least busy host. Jobs are not moved to oneself.

Algorithm 2: Each entity compares its own busyness to its observation (estimate) of the busyness of every other host. All differences less than or equal to bias1 imply no jobs are moved to those hosts. If the difference is greater than bias1 but less than bias2 then one job is moved there. If the difference is greater than or equal to bias2 then two jobs are moved there. In no case are more than (y)(z) jobs moved at one time from a host, where y = fraction of jobs permitted to be moved, and z = number of jobs currently at this host. If there is more demand for jobs than an entity is permitted to move, it satisfies the demand in a pre-determined fixed order.

Algorithm 3: This algorithm performs in the same way as algorithm 1 except when an entity, e_i, sends a job to host k at time t, it records this fact. Then for time delta t, called a window, this entity will not send any more work to host k. If at any time during the window period, entity e_i calculates that host k is least busy then no job is moved during such an activation. Of course, during the window, jobs may be sent to other hosts if they are observed as least busy by greater than the bias (same bias as in algorithm 1).

One prime motivation behind these three algorithms is that they are all very simple and inexpensive to run, necessary conditions for job scheduling algorithms. In algorithm 1 the relative busyness between host i and the least busy host (plus a bias) is used to determine if a job should move. This is about the simplest algorithm we can devise. Since algorithm 1 only moves jobs to the least busy host we felt that a better algorithm might be to spread the work around, i.e., move some work to all the lightly loaded hosts from the heavily loaded ones. Hence algorithm 2 was devised. Finally, we were worried that jobs in transit to a lightly

loaded host were not taken into account possibly producing instabilities. For example, if (1) host 1 was very busy, (2) host 5 was least busy, (3) host 1 were activated every 2 seconds, and (4) it took 16 seconds for jobs to reach host 5, then host 1 could conceivably send at least 8 jobs to host 5 before the first one was received. Other hosts could be doing the same thing. This could result in an unstable situation. Algorithm 3 was designed to avoid such problems.

5.0 SIMULATION AND ANALYTICAL RESULTS

This section presents the simulation results, compares the simulation results to analytical results, and discusses the implications of the results.

In these simulations four main characteristics were studied:

1. Parameters of the Algorithms - each algorithm has one or more parameters called biases in this paper (see section 4).

2. Arrival Rates - Four different sets of arrival rates are used in the simulations (Table 1). The sets of arrival rates are labeled tuning, light, moderate, and heavy. In tuning the three algorithms, it was decided to use a different set of arrival rates than those used for the subsequent algorithm comparisons. This is because, in practice, such algorithms would not be tuned precisely for the current arrival rates, but would be only an approximation. The light and moderate loads are considered the normal network situation. It was also decided to simulate a heavy system load to determine how the algorithms perform when, for relatively short times (40 minutes), a system experiences arrival rates greater than its ability to service them.

3. Delay in the subnet - The change in the amount of traffic in the subnet affects response times and load balancing. The affect on these algorithms is studied here.

4. Scheduling Interval - The affect of a change in the scheduling interval from 2 -> 4 -> 8 -> 16 seconds is tested.

While many other characteristics could be studied including size of the network, topology, speeds of the hosts, etc. we felt that the characteristics studied here are some of the most important.

Table 1 - Arrival Rates (Jobs/Second)

HOST	TUNING	LIGHT	MODERATE	HEAVY
1	.18	.1176	.153	.2
2	.1	.1	.125	.2
3	.111	.111	.143	.2
4	.133	.0588	.143	.2
5	.125	.125	.125	.2

5.1 Analytical Results - No Network

Before the network simulation model was run, it was decided to obtain an upper bound on response time for the system. Therefore, the five hosts of our model were treated as independent M/M/1 queues for each set of arrival rates with no job scheduling algorithm and no network. If a system-wide job scheduling algorithm cannot do better than leaving all incoming jobs at their site of arrival, then as far as performance is concerned the algorithm is useless. The response times (wait time plus execution time) are 37.4 seconds for the tunning arrival rates, 29.22 seconds for the light load arrival rates, and 44.67 seconds for the moderate load arrival rates. One cannot apply the M/M/1 results to the heavy load arrival rates because the arrival rates are faster than the service rates implying infinite queues. Heavy arrival rates are used only to test the operation of the algorithm assuming very heavy loads would exist for a short period. Fortunately, as will be shown, all system-wide algorithms presented in this paper perform considerably better than the no network situation.

5.2 Simulation Results - Network With Imperfect Knowledge

Algorithms 1, 2 and 3 were tuned and then compared for light, moderate, and heavy system loads. Also tested was the effect of various delays in the subnet and scheduling intervals. The parameters of the tuning runs are identical to all other runs except for arrival rates and the specific parameters (described below) being tuned. In all the tables below response times are given in seconds with 90% confidence intervals, and load balancing statistics are given in average number of jobs per host over three runs. Movement cost is given in percentage of job movement with 90% confidence intervals. By percentage of movement is meant the total number of jobs moved divided by the total number that entered the system. Note that jobs may move more that once and each move is counted separately because each move adds overhead to the system. The tuning arrival rates were chosen as an approximate mixture of light and moderate arrival rates but

biased more towards the moderate arrival rates.

5.2.1 Tuning -

Tuning algorithm 1 means choosing the proper bias between the number of jobs at the current host and the number at the least busy host. A bias equal to 2 was chosen for all subsequent runs because it provides a combination of good response time (an average of 14.7 sec) and load balancing with reasonably low job movement (an average of 27.6%). See the column under bias=2 in Table 2. For a bias=0 the variation in the % of job movement was somewhat unstable as shown by the 90% confidence interval (75±28.5%). Percentage of movement greater than 100% is possible because jobs can move more than once. This implies that with the wrong bias algorithm 1 may not be very stable.

Table 2 - Algorithm 1 Tuning

	BIASES			
	BIAS=0	BIAS=1	BIAS=2	BIAS=3
Response Time	14.13±.15	14.54±.42	14.7±.37	16.54±.7
% of Movement	75±28.5	47.3±10.7	27.6±1.45	24.3±2.5
Load Balancing (av over 3 runs)				
Host 1	1.06	1.37	1.67	1.80
Host 2	1.28	1.06	1.05	1.50
Host 3	0.86	1.05	0.96	1.34
Host 4	0.85	0.77	0.82	1.31
Host 5	0.82	1.30	1.12	1.29

Tuning algorithm 2 means choosing bias1, bias2, and the fraction y of jobs permitted to be moved (refer back to section 4). Too many simulations would be required to tune all the possible combinations of these three biases. Educated guesses were made in the following way. It was argued that since there are only four hosts, excluding oneself, we would not move more than 3 at any one time (y bias). This allows movement to a large percentage of other hosts but in a controlled way. A number of preliminary simulation runs showed that only infrequently was the difference between two host's busyness greater than 6 (about 4% of the time). We chose bias2 = 6 so that 2 jobs moved to a host at one time would occur infrequently, but would occur. At this point

bias1 was tested and the results appear in Table 3. The result is that bias1 = 3 seems to work best and was used in subsequent testing.

Table 3 - Algorithm 2 Tuning

	BIAS1		
	BIAS1=2	BIAS1=3	BIAS1=4
Response Time	24.1±.45	16.3±.61	19.9±.55
% of Movement	110±33	25±4.3	23±3.6
Load Balancing (av over 3 runs)			
Host 1	1.50	1.21	1.63
Host 2	1.49	1.27	1.80
Host 3	1.29	1.15	1.93
Host 4	1.09	0.87	1.53
Host 5	1.05	0.98	1.61

Tuning algorithm 3 required picking the right bias and window. Since algorithm 3 is so similar to algorithm 1, and since bias=2 worked so well for algorithm 1, it was decided to use the same bias for algorithm 3. A window size of 2 seconds produced the best results (Table 4). For the window size of 2 seconds there is good response time (an average of 14.69 sec), good load balancing, and a reasonably small percentage of jobs moved (an average of 16.6%). Notice that choosing a window size of 6 seconds degrades response time (an average of 19.83 sec), does not balance the load, and only a small percentage of jobs moved (an average of 13.1%). This indicates that the window size was too large to enable effective performance improvement. A window size of 8 seconds performed proportionally worse (Table 4).

Table 4 - Algorithm 3 Tuning

	WINDOW SIZE			
	1 sec	2 sec	6 sec	8 sec
Response Time	22.21±.53	14.69±.47	19.83±.23	21.57±.61
% of Movement	31±1.9	16.6±1.7	13.1±1.15	22.33±1.4
Load Balancing (av over 3 runs)				
Host 1	6.01	1.95	4.88	3.59
Host 2	1.71	0.88	0.96	1.84
Host 3	1.22	1.19	0.89	1.30
Host 4	1.06	0.83	0.91	1.26
Host 5	0.85	0.85	1.02	2.24

5.2.2 Vary Arrival Rates -

Tables 5, 6 and 7 show the results of the simulation runs by comparing algorithms 1, 2 and 3 for light, moderate, and heavy loads when the average delay in the subnet is approximately 8 seconds per job. In general, algorithms 1 and 3 perform similarly in terms of response time and load balancing, but algorithm 3 moves less jobs in achieving the result. For example, in the moderate load case (Table 6) algorithm 3's percentage of jobs moved is only an average of 40.75% as compared to algorithm 1's average of 57.5%, but its response time is slightly worse (an average of 24.02 sec as compared to an average of 22.81 sec).

On the other hand, algorithm 2 does not perform quite as well as algorithm 1 and 3 on light loads, significantly better on moderate loads, and worse on heavy loads. For light loads, having each host compare itself to all other hosts causes too much job movement and therefore produces slightly worse performance than algorithms 1 and 3. For moderate loads algorithm 2's spreading work to more than 1 lightly loaded host does pay dividends in the sense of improved response times. The poor performance of algorithm 2 under heavy loads is due to the lack of a cutoff. Notice (in Table 7) that algorithm 2 continues to do load balancing regardless of the system load causing a tremendous amount of useless job movement (an average of 118%).

AD-A132 893 DDCENTRALIZED CONTROL OF SCHEDULING IN DISTRIBUTED SYSTEMS(U) MASSACHUSETTS UNIV AMHERST DEPT OF ELECTRICAL AND COMPUTER ENGINEERING J A STANKOVIC 2/2

UNCLASSIFIED 18 MAR 83 DAAB07-82-K-J015 F/G 9/2 NL

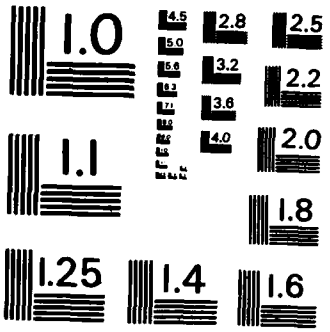


Table 5 - Light Load Comparison

	ALGORITHM 1	2	3
Response Time	14.2±.41	15.2±2.2	14.9±.78
% of Movement	1	19.9±7.5	14.8±.4
Load balancing (av over 3 runs)			
Host 1	0.85	0.77	0.77
Host 2	1.03	0.92	1.17
Host 3	0.83	0.77	0.84
Host 4	0.15	0.29	0.27
Host 5	0.94	1.23	1.35

Table 6 - Moderate Load Comparison

	ALGORITHM 1	2	3
Response Time	22.81±.1	17.5±2.5	24.02±1.5
% of Movement	57.5±3.1	36±3.1	40.75±4.7
Load Balancing (av over 3 runs)			
Host 1	1.66	1.41	1.58
Host 2	1.66	1.53	2.07
Host 3	1.73	1.35	2.82
Host 4	1.43	1.01	1.83
Host 5	1.52	1.33	2.16

Table 7 - Heavy Load Comparison

	ALGORITHM		
	1	2	3
Response Time	330.1±40.6	345.2±17	350±70
% Of Movement	28.5±2.2	118±5	7±5.1
Load Balancing (av over 3 runs)			
Host 1	54.0	50.7	44.4
Host 2	85.5	51.2	66.5
Host 3	69.0	50.9	54.5
Host 4	57.5	50.6	39.8
Host 5	104.0	50.9	55.1

5.2.3 Vary Subnet Delay -

Next, consider the effect of different average delays (4, 8, 16 and 24 seconds) for jobs moving through the subnet. Tables 8, 9 and 10 present the response time and movement statistics for the three algorithms under light and moderate loads. Load balancing statistics are not shown because they do not provide any additional information over that given in Tables 5, 6 and 7.

Table 8 - Algorithm 1 Varying Subnet Delays

	SUBNET DELAYS (sec)			
	4	8	16	24
LIGHT LOAD				
Response Time	12.7±.3	14.2±.41	15.04±.7	16.4±.63
% of Movement	10±2	11±.5	15.3±.9	8.2±1.7
MODERATE LOAD				
Response Time	17.1±1.3	22.8±.1	27.7±1	42.04±4.6
% of Movement	35±2.6	57.5±3.1	62.6±4.1	91±11.0

Table 9 - Algorithm 2 Varying Subnet Delay

	SUBNET DELAYS (sec)			
	4	8	16	24
LIGHT LOAD				
Response Time	14.1±.8	15.2±2.2	17.0±1.9	18.8±.7
% of Movement	14.4±1.1	19.9±7.5	21.7±4.3	22.2±2.1
MODERATE LOAD				
Response Time	18.6±1.7	17.5±2.5	23.7±1.6	25.7±2.1
% Of Movement	31.5±4.8	36±3.1	45±4	43±6.7

Table 10 - Algorithm 3 Varying Subnet Delay

	SUBNET DELAYS (sec)			
	4	8	16	24
LIGHT LOAD				
Response Time	14.0±.3	14.9±.78	14.8±.3	17.5±.61
% of Movement	9.6±.4	14.8±.4	15±1.1	15.01±1.6
MODERATE LOAD				
Response Time	22.4±.7	24.02±1.5	33.4±3.2	42.8±3.9
% of Movement	27±1.7	40.75±4.7	57.9±2.0	70±3.0

The results presented in Tables 8, 9 and 10 show increased response time with increased delay in the subnet as is to be expected. However, there are two interacting factors that are involved. As the average delay for moving jobs through the subnet grows, there is greater cost in moving a job. Initially one might conclude that this adds to the average delay for a job in the system. Although this occurs sometimes, it is not necessarily true. A job moving through the subnet is doing so, presumably, to arrive at a shorter queue (in time) than the one it just left. Hence, only if the delay in the final queue (where it actually receives service) plus the time needed to move is greater than the delay it would have experienced without moving or by moving faster, is there a net degradation in system response time. It seems likely that there is a tradeoff point that is some function of

the delay in the subnet and the waiting times in the hosts. This is the first factor affecting the results presented in Tables 8, 9 and 10.

The second factor involves a host's view of the network. When host i sends a job to host k, it's queue length is decreased by one. Host k queue length is not increased until some later time. Jobs in the subnet are no longer in any queue, therefore, temporarily out of the system in the host's eyes. In some situations (when there are alot of jobs in the subnet), there are fewer candidates for job movement and, therefore on the average less jobs move. Having less candidates for movement may give rise to increased response times.

The results seem to indicate that as delay in the subnet grows, job movement increases and response time decreases (gets worse). The magnitude of the changes being smaller than expected are believed to be caused by a combination of the above described factors.

5.2.4 Vary Scheduling Interval -

Since algorithm 2 performed the best we decided to first study the effect of varying the scheduling interval on algorithm 2. The results are shown in Table 11. Notice that there is little difference in response time for scheduling intervals of 2, 4, and 8 seconds. Hence, there is no need to run the algorithm faster than every 8 seconds. When the scheduling interval is 16 seconds one starts to see a degradation in response time because the reaction time is not fast enough and there is not enough job movement to produce a good response time. Several similar runs were also made with algorithms 1 and 3 and they produced similar results. Rather than making alot more runs to generate confidence intervals for algorithms 1 and 3 we decided to combine varying the scheduling interval with a large subnet delay.

Table 12 shows the results of varying the scheduling interval when the delay in the subnet is large (16 sec). These results are interesting in that the smallest scheduling interval results in poorest performance. For example, when the scheduling interval is 2 seconds, the response time is on the average 33.4 seconds and there is an average of 57.9% job movement. In this case too many jobs are moved into the subnet where the delay is now very substantial. Therefore, when the cost of moving a job gets too great, it pays to move fewer jobs as is the case when the scheduling interval is 8 seconds. This is seen in Table 12 where only an average of 16.8% of the jobs move resulting in better response time because the delay in the subnet is dominating the response time of the system. To mitigate this problem a possible variation to algorithm 3 would be to adapt the window size

(currently 2 seconds) based on the current estimated delay in the subnet.

Table 11: Algorithm 2 Scheduling Interval

(delay in subnet - 8 sec)

	Scheduling Interval (sec)			
	2	4	8	16
MODERATE LOAD				
Response Time	17.5±2.5	17.6±2.2	18.9±3.1	20.8±2.7
% of Movement	36±3.1	31.6±7.2	23.4±2.5	19±1.8

Table 12: Algorithm 3 Scheduling Interval

(delay in subnet - 16 sec)

	Scheduling Interval (sec)		
	2	4	8
MODERATE LOAD			
Response Time	33.4±3.2	27.5±3.4	22.0±3.9
% of Movement	57.9±2	36.9±5.1	16.9±2.8

5.3 Analytical Results - Fractional Assignment

Assume for a moment that each host knows the fraction of jobs arriving at its site from the external world that it should keep and the fraction that it should send to each of the other hosts to minimize response time. In other words assume that the statistical properties of the network were known. Such an algorithm would have lower cost than the three algorithms just evaluated because no state information is passed around the network. But how would this "fractional assignment" algorithm compare to the above algorithms as far as response time is concerned. By making some simplifying assumptions, a queueing model can be formed and solved for this fractional assignment algorithm.

Assume a central queue with overall arrival rate $\lambda = \sum \lambda_i$, and 5 hosts with service rates $\mu_1, \mu_2, \mu_3, \mu_4, \mu_5$. When jobs enter the central queue they are immediately

transferred to one of the hosts based on the optimal fractional assignment, f_i, to each host (Figure 3) to minimize response time.

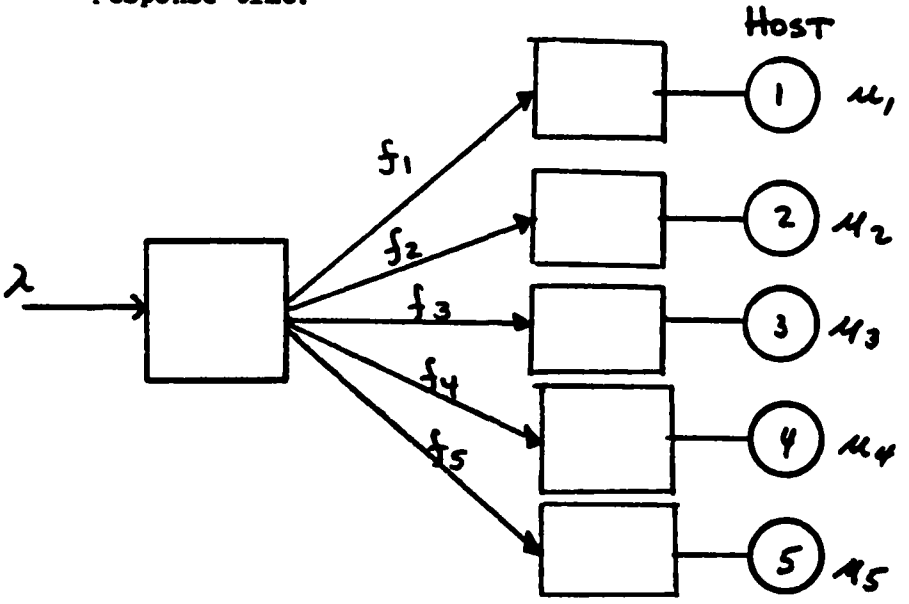


Figure 3: Fractional Assignment Queueing Model

The system response time, T, is given by

$$T = \sum_{i=1}^{5} f_i / (\mu_i - \lambda f_i),$$

and the fraction f_i can be calculated from

$$f_i = \frac{\mu_i}{\lambda} - \frac{(\sqrt{\mu_i})(\sum_{i=1}^{5} \mu_i - \lambda)}{\lambda \sum_{i=1}^{5} \sqrt{\mu_i}}.$$

For a derivation of the formula for f_i see the Appendix. The results of these calculations for the values used in the simulations are given below in Table 13.

Table 13 - Fractional Assignment Response Times

	System Load	
OPTIMUM FRACTION	LIGHT	MODERATE
.24 .16 .20 .24 .16	14.59	30.55

Then, using the conservation of flow principle and $\sum_j P(j) = 1$, we obtain

$$P_0 = 1/\left(\sum_{J=0}^{N-1} \lambda^j / M(j) + \frac{\lambda^N}{M(N)} / (1 - \lambda/\mu(N)) \right).$$

The expected queue length is given by

$$\bar{L} = P_0 \left(\sum_{j=1}^{N-1} j \frac{\lambda^j}{M(j)} + \frac{\lambda^N}{M(N)} \left[\frac{N-1}{1-\lambda/\mu(N)} + (1-\lambda/\mu(N))^{-2} \right] \right).$$

Using Little's formula, the expected delay is

$$T = \bar{L}/\lambda .$$

The response times, T, calculated as lower bounds, are given in Table 14.

Table 14 - Lower Bound Response Times

System Load	
Light	Moderate
6.16 sec	9.3 sec

This lower bound calculation assumes no communication subnet delay, and that a job can be preempted and moved to a faster processor as soon as that processor becomes available, again at no cost. Although this analytical model cannot be achieved in practice, it does serve to put a lower bound on our results.

5.5 Comparison And Discussion

For easy comparison Tables 15 and 16 summarize some of the statistics presented in some of the previous tables.

For light loads (Table 15), algorithms 1 and 3 are about equal, improving the no network response time by about 50% at a cost of moving an average of between 11 and 14.8 percent of their jobs, respectively. Both of these algorithms also peform slightly better than the fractional assignment algorithm when you consider that the response time calculated for it did not include delay due to job movement.

At moderate loads (Table 16), algorithms 1 and 3 improve the no network response times by approximately 49% and the fractional assignment response times by about 25%. Furthermore, for moderate loads algorithm 3 is superior to algorithm 1 because it achieves approximately the same performance (response time and load balancing) with less cost (job movement). Although such costs are already figured into

Note that the response times calculated here do not contain any delay for actual movement of jobs so this is a very optimistic figure. Further, it was assumed that the arrival rates were known so that the f_i's are the best possible fractions. Nevertheless, at a moderate load the three algorithms above making use of state information, perform considerably better than the fractional assignment. At light loads, they seem to be about equal.

5.4 Analytical Results - Lower Bound

The previous results indicate that very simple decentralized system-wide job scheduling algorithms can improve response time considerably. However, in theory, how much better can we expect to do, i.e., what is the lower bound. Analytical solutions to the complex network situation we simulated are not known. However, by making simplifying assumptions it is possible to analytically obtain some idea of the lower bound.

Assume that the arrival process is Poisson and that service times are described by an exponential distribution. Assume that the service rate μ_i is ordered so that $\mu_i \geq \mu_j$ iff $i <= j$. For our network of 5 hosts, this situation is described by the state transition diagram given in Figure 4.

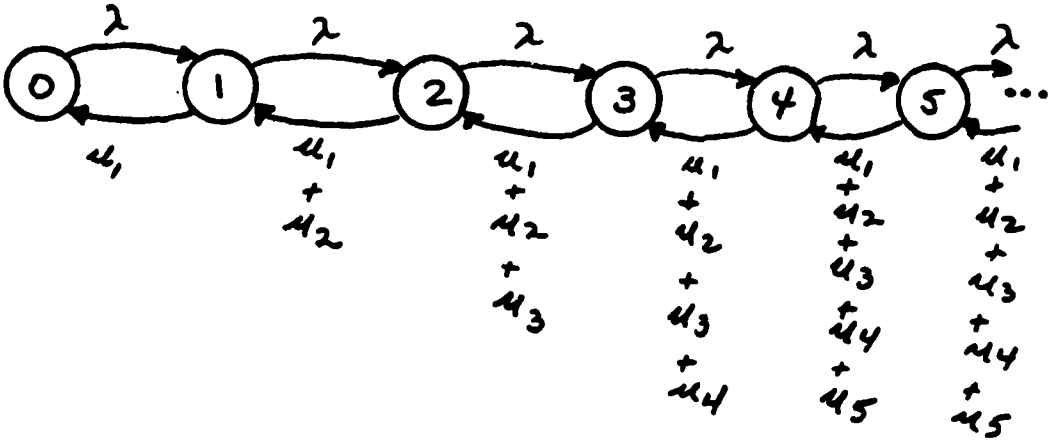


Figure 4 - State Transition Diagram

In general, let the number of processors be N. Define

$$u(i) = \sum_{j=1}^{i} u_j,$$

$$M(i) = \prod_{j=1}^{i} u(i) \qquad i = 1, 2, \ldots,$$

$$M(0) = 1.$$

response times we consider algorithm 3 better than algorithm 1 because we wish to avoid unnecessary job movement.

Note that for both light and moderate loads algorithms 1 and 3 perform better than the fractional assignment algorithm because they are able to adapt to the information about the state of the network in an effective way rather than relying on statistical properties. In conclusion, algorithms 1 and 3 seem to be stable and significantly improve performance, i.e., multiple decentralized controllers of a stochastic replicated function are cooperating effectively at very low cost. However, without good tuning such algorithms are also susceptible to instabilies. This implies that one might need to dynamically adapt the parameters of the algorithms to maintain stability over widely divergent conditions. On the other hand our simulations indicate that the tuned parameters are robust over reasonably different arrival rates.

Overall, algorithm 2 performs even better than algorithms 1 and 3 but has a tendency to move too many jobs if not tuned well (see the tuning runs). The tuning is also more complicated because of the three biases, but as for algorithms 1 and 3 the chosen biases seem to be fairly robust over the various arrival rates. This implies that the algorithm can retain its simplicity and operate over a wide range of arrival rates. A modification would be to dynamically adapt the parameters (biases) of the algorithm paying the associated cost. Such a modification could be the basis for a future study. For moderate loads tne response time of algorithm 2 was 61% better than the no network case, 43% better than the fractional assignment case, and approximately 24% better than algorithms 1 and 3. We, therefore, recommend algorithm 2 but cutoffs, such as those found in algorithms 1 and 3 should be added to algorithm 2.

Table 15: Summary Statistics - Light Load

(Averages only - 90% CI shown in previous tables)

	No Network (Upper Bnd)	Network Simulation A1	A2	A3	Fractional Assign.	Lower Bound
Response Time	29.22	14.2	15.2	14.9	14.6	6.16
Movement	0	11	19.9	14.8	-	-
Load Balancing						
Host 1	-	0.85	0.77	0.77	-	-
Host 2	-	1.03	0.92	1.17	-	-
Host 3	-	0.83	0.77	0.84	-	-
Host 4	-	0.15	0.29	0.27	-	-
Host 5	-	0.94	1.23	1.35	-	-

Table 16: Summary Statistics - Moderate Load

(Averages only - 90% CI shown in previous tables)

	No Network (Upper Bnd)	Network Simulation A1	A2	A3	Fractional Assign.	Lower Bound
Response Time	44.67	22.81	17.5	24.02	30.55	9.3
Movement	0	57.5	36	40.7	-	-
Load Balancing						
Host 1	-	1.66	1.41	1.58	-	-
Host 2	-	1.66	1.53	2.07	-	-
Host 3	-	1.73	1.35	2.82	-	-
Host 4	-	1.43	1.01	1.83	-	-
Host 5	-	1.52	1.33	2.16	-	-

6.0 SUMMARY

This paper presents the simulation results of three stochastic, replicated, decentralized controlled job scheduling algorithms operating in a network with imperfect knowledge. It was shown that the multiple controllers of such algorithms can effectively cooperate with each other to achieve a system-wide objective of good response time and load

balancing with limited job movement. When tuned properly the algorithms operated in a stable manner. The tuning was shown to be necessary but once tuned the algorithms were fairly robust for different arrival rates. A suggested modification is to dynamically adapt the parameters of the algorithms but the added cost of such a scheme must also be addressed.

By comparing performance of the three algorithms to each other and to analytical results, it was concluded that while algorithms 1 and 3 worked well, algorithm 2 was the best. An important result is that very simple (execute fast) decentralized controlled job scheduling algorithms can effectively improve performance. Another result is that by utilizing a minimal amount of state information (number of jobs) it was shown that one gets better performance than the optimal fractional assignment which we obtained analytically. Another modification that we suggest is to avoid moving very large (in size) jobs because this congests the subnet and degrades overall response time. This additional state information is easy to obtain and the added execution time checks required are trivial. In summary, we feel the results presented here provide some insight into the performance and stability of decentralized control job scheduling algorithms as part of distributed processing systems.

7.0 ACKNOWLEDGMENTS

I would like to thank Don Towsley for his suggestion to compare the simulation results to the best "fractional assignment" and for his help with the analytical models.

8.0 APPENDIX

This appendix describes the derivation of the formula for f_i given in section 5.3. Starting with

$$T = \sum_{i=1}^{5} f_i/(\mu_i - \lambda f_i),$$

we need to minimize delay, T, subject to $\sum_{i=1}^{5} f_i = 1$. The Lagrangian function can be defined as

$$L = \sum_{i=1}^{5} \frac{f_i}{\mu_i - \lambda f_i} - g\left(\sum_{i=1}^{5} f_i - 1\right)$$

where g is the Lagrangian multiplier. Taking the derivative and setting it equal to zero results in

$$\frac{\partial L}{\partial f_i} = \frac{\mu_i}{(\mu_i - \lambda f_i)^2} - g = 0$$

$$f_i = \left(\mu_i - \frac{\sqrt{\mu_i}}{\sqrt{g}}\right)/\lambda \qquad (1)$$

Since $\sum_{i=1}^{5} f_i = 1$ we have

$$\sum_{i=1}^{5} \mu_i/\lambda - \sum_{i=1}^{5} \sqrt{\mu_i}/\lambda\sqrt{g} = 1$$

$$\sqrt{g} = \sum_{i=1}^{5} \sqrt{\mu_i} / \left(\sum_{i=1}^{5} \mu_i - \lambda \right)$$

Substituting $\sqrt{g}$ into formula (1) gives

$$f_i = \frac{\mu_i}{\lambda} - \frac{(\sqrt{\mu_i})\left(\sum_{i=1}^{5} \mu_i - \lambda \right)}{\lambda \sum_{i=1}^{5} \sqrt{\mu_i}} .$$

9.0 REFERENCES

[1] Agrawala, A. K., S. K. Tripathi, and G. Ricart, "Adaptive Routing Using a Virtual Waiting Time Technique," IEEE Transactions on Software Engineering, Vol. SE-8, No. 1, January 1982.

[2] Aoki, Masanao, "Control of Large-Scale Dynamic Systems by Aggregation," IEEE Transactions on Automatic Control, June 1978.

[3] Bokhari, S. H., "Dual Processor Scheduling with Dynamic Reassignment," IEEE Transactions on Software Engineering, Vol. SE-5, No. 4, July 1979.

[4] Bryant, R. M. and R. A. Finkel, "A Stable Distributed Scheduling Algorithm," Proceedings 2ND International Conference in Distributed Computing Systems, April 1981.

[5] Chow, Yuan-Chieh, and Walter Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," IEEE Transactions on Computers Vol C-28, No. 5, May 1979.

[6] Chu, W. W., "Optimal File Allocation in a Multiple Computing System," IEEE Transactions on Computers, Vol. C-18, pp 885-889, October 1969.

[7] Chu, W. W., L. J. Holloway, M. Lan, and K. Efe, "Task Allocation in Distributed Data Processing," IEEE Computer, Vol. 13, pp 57-69, November 1980.

[8] Efe, Kemal, "Heuristic Models of Task Assignment Scheduling in Distributed Systems, IEEE Computer, Vol. 15, No. 6, June 1982.

[9] El-Dessouki, O. I., and W. H. Huan, "Distributed Enumeration on Network Computers," IEEE Transactions on Computers, Vol. c-29, pp 818-825, September 1980.

[10] Enslow, Philip, "What is a Distributed Data Processing System," IEEE Computer, Vol. 11 No. 1, January 1978.

[11] Farber, D. J. , et al, "The Distributed Computer System," Proceedings 7th Annual IEEE Computer Society International Conference, February 1973.

[12] Gonzalez, M. J., "Deterministic Processor Scheduling," ACM Computing Surveys, Vol. 9, No. 3, pp 173-204, September 1977.

[13] Jarvis, R. A., "Optimization Strategies in Adaptive Control: A Selective Survey," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-5, No. 1, January 1975.

[14] Jensen, Douglas, "The Honeywell Experimental Distributed Processor -- An Overview of Its Objectives, Philosophy and Architectural Facilities," IEEE Computer, Vol. 11, No. 1, January 1978.

[15] Kleinrock, L., and A. Nilsson, "On Optimal Scheduling Algorithms for Time-Shared Systems," JACM, Vol. 28, No. 3, pp 477-486, July 1981.

[16] Larsen, R. E., "Tutorial: Distributed Control," IEEE Catalog No. EHO 153-7, IEEE Press, New York, 1979.

[17] LeLann, G., "Distributed Systems - Towards a Formal Approach," Proceedings IFIP Congress, Toronto, North Holland Publ., pp 155-160, August 1980.

[18] Ma, P. Y. R., E. Y. S. Lee, and M. Tsuchiya, "A Task Allocation Model for Distributed Computing Systems," IEEE Transactions on Computers, Vol. C-31, No. 1, pp 41-47, January 1982.

[19] Ousterhout, John K., D. Scelza, and P. Sindu, "Medusa: An Experiment in Distributed Operating System Structure," CACM, Vol. 23, No. 2, February 1980.

[20] Sandell, Nils R., Pravin Varaiya, Michael Athans, and Michael Safonov, "Survey of Decentralized Control Methods for Large Scale Systems, IEEE Transactions on Automatic Control, Vol. AC-23, No. 2, April 1978.

[21] Smith, R. G., "The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver," IEEE Transactions on Computers," Vol. C-29, No. 12, Dec. 1980.

[22] Stankovic, John A., "A Comprehensive Framework for Evaluating Decentralized Control," Proceedings 1980 International Conference on Parallel Processing, August 1980.

[23] Stankovic, John A., "Analysis of a Decentralized Control Algorithm For Job Scheduling Utilizing Bayesian Decision Theory," Proceedings 1981 International Conference on Parallel Processing, August 1981.

[24] Stankovic, John A., N. Chowdhury, R. Mirchandaney, I. Sidhu, "An Evaluation of the Applicability of Different Mathematical Approaches to the Analysis of Decentralized Control Problems," Proceedings COMPSAC82 November 1982.

[25] Stankovic, John A., "A Stochastic Model For Replicated Decentralized Control," submitted to IEEE Transactions on Computers, Dec. 1982.

[26] Stankovic, John A., "A Heuristic for Cooperation Among Decentralized Controllers," Proceedings INFOCOM 83, April 1983.

[27] Stankovic, John A., "Bayesian Decision Theory and Its Application to Decentralized Control Algorithms," in preparation.

[28] Stone, H. S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, pp 85-93, January 1977.

[29] Stone, H. S., and S. H. Bokhari, "Control of Distributed Processes," IEEE Computer, pp 97-106, July 1978.

[30] Tenney, Robert R., and Nils R. Sandell, Jr., "Structures for Distributed Decisionmaking," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, No. 8, pp 517-527, August 1981.

[31] Tenney, Robert R., and Nils R. Sandell, Jr., "Strategies for Distributed Decisionmaking," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, No. 8, pp 527-538, August 1981.

[32] Wittie, L., and Andre M. van Tilborg, "MICROS, A Distributed Operating System for Micronet, A Reconfigurable Network Computer, IEEE Transactions on Computers," Vol. C-29, No. 12, December 1980.

DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2 copies
Commander USAERADCOM ATTN: DELSD-L-S Fort Monmouth, N.J. 07703	2 copies
Commander USACECOM ATTN: DRSEL-COM-RF (Dr. Klein) Fort Monmouth, N.J. 07703	1 copy
Comander USACECOM ATTN: DRSEL-COM-D (E. Famolari) Fort Monmouth, N.J. 07703	1 copy
Comander USACECOM ATTN: DRSEL-COM-RF-2 (C. Graff) Fort Monmouth, N.J. 07703	10 copies

END

FILMED

12-85

DTIC